Eg:- $(16)_{10} = (100)_b$. Find b.

Solve:-

$16 = 1 \times b^2 + 0 \times b + 0 \times b^0$.

$\Rightarrow b = 4$.

Eg: $(292)_{10} = (1204)_b$

Solve:-

$292 = 1 \times b^3 + 2b^2 + 4$

$\Rightarrow b^3 + 2b^2 - 288 = 0$.

$\Rightarrow$ **b = 6 (By Hit & Trial)**

Binary Codes

A binary code is just an assignment of information to bit patterns.

- Weighted Codes
  - Eg:- BCD.
- Non-weighted codes.
  - Eg:- GRAY CODE, Excess-3 code

(1) BCD : (Also called 8421 code)

- formed by converting each digit of a decimal number into binary.

Eg:-

$(25)_{10} = (11001)_2 = (\underset{2}{0010}\ \underset{5}{0101})_{BCD}$

- requires more digits than conventional binary.

Que:- How many bits are required for $0 < \quad < 999$.

(i) Binary $\Rightarrow 2^{10} = 1024 \Rightarrow$ 10 bits

(ii) BCD $\Rightarrow$ 999 $\Rightarrow$ 4+4+4 = **12 bits**

(4 4 4)

* 10-15 are invalid combinations in BCD.
  0-9 are valid only.

Eg:- $(15)_{10} = (1111)_2 = (0001\ 0101)_{BCD}$

BCD Addition:

For sum ≤ 9.

5	0101		5	0101
+3	+ 0011		+5	0101
8	1000	00010000 ⇐	10	1010
		Valid		Invalid

8	1000		15	0001 0101
+9	1001		+17	0001 0111
17	10001		32	0010 1100 → >9
	0110 Add six.		+	0110 (Add six)
	1 0111 → 1 7			0011 0010 → 3 2

(2) 2421 code:-

- Represents from 0 to 9.
- Self complementing code.

Eg:- $(6)_{10} = (1100)_{2421}$

9' compliment of 6 = 3 ⟹ $(0011)_2$

↓ compliment

$(1100)_{2421}$

Non - weighted codes:-

(1) Excess 3 code:- (XS-3)

- Self complementing code.

~~(2) GRAY code~~ :-

Eg:- $(643)_{10}$

↓ Add 3 to each bit

(976) → (1001 0111 0110) Excess-3.

Eg:- $(1001\ 0111\ 0110)_{XS-3}$ →

↓

976

↓ Subtract 3 from each bits.

$(643)_{10}$

Scanned by CamScanner

(2) GRAY CODE :-

- Every transition from one value to the next value involves only one bit change.
- Good for error detection.
- Also called cyclic / reflected code.

$G_3 = B_3$

$G_2 = B_3 \oplus B_2$

$G_1 = B_2 \oplus B_1$

$G_0 = B_1 \oplus B_0$.

Eg:- Binary to GRAY CODE

$(10.1101)_2 = (11.1011)_G$

---

* Alphanumeric Codes

- Alphabets
- '0-9'
- Special Symbols
- SOH, NULL, BELL (Non-printable).

Eg:- (1) ASCII → Previously 7 bit code
→ 8 bit code at present

(2) EBCDIC (8-bit) → 8 bit code.

Parity

- Method used for error detection.

Event parity : No. of 1's must be even

Odd Parity : No. of 1's must be odd.

Eg :-

	Even Parity	Odd Parity
11001	110011	110010
11110	111100	111101

Binary Number System

Unsigned Binary code

Range = $[0, 2^n - 1]$

No. of bits = n bits

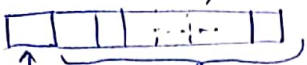
Eg:- $(26)_{10} = (11010)_2$

Using 8 bits :-

$(26)_{10} = (00011010)_2$

- ARITHMETIC Operations:
  - → Addition
  - → Subtraction
  - → multiplication
  - → Division.
- Logic Operations:
  - → AND (Conjuction)
  - → OR (Disjunction)
  - → NOT (Negation)
  - → XOR (Exclusive Or)

Signed Binary Code



Signed bits (+ve = 0, -ve = 1) | magnitude bits

- Used to represent +ve & -ve numbers.
- Three types :-
  - Signed magnitude
  - 1's compliment
  - 2's compliment.

* All three above are same for +ve numbers, but different for -ve numbers.

* Range = $[-2^n, 2^n - 1]$

Floating point System.

Eg:- $(1010100)_2 - (1000100)_2$

Solve:-

2's comp. of $(1000100)_2 = (0111100)_2$

So,
```
  1010100
  0111100
---------
 10010000
```
Ans:- 0010000

Eg:- $(1000100)_2 - (1010100)_2$

↓ 2's compliment

```
  0101011
+       1
---------
  0101100
```

```
  1000100
+ 0101100
---------
  1110000
```

↓ 2's compliment

```
  0001111
+       1
---------
 -0010000
```

Answer:- -0010000

*Read 2's & 1's compliment again.

Eg:- $(1100)_2 - (1100)_2$

Solution:-

↓ 1's compliment

0011

```
  1100
+ 0011
------
  1111
```
1's compliment → $-(0000)$

Drawback of 1's compliment: we can get +ve & -ve (two types of) zeroes.

That's why, 2's compliment method is advantageous over 1's compliment method.

Signed Binary Codes:- Using 8-bits

↓

	+9	-9	-8
Signed magnitude	00001001	10001001	
1's compliment	00001001	11110110	
2's compliment	00001001	11110111	

1's compliment Arithmetic.

Eg:- $(6)_{10} + (9)_{10}$ using 1's compliment.

$+6 \longrightarrow 0\ 0000110$

$+9 \longrightarrow 0\ 0001001$

$15 \qquad\quad 0\ 0001111$

Eg:-

$-6 \longrightarrow 1\ 1111001$

$+9 \longrightarrow 0\ 0001001$

$3 \qquad\quad \boxed{1}\ 0\ 0000010$

$+ \longrightarrow 1$

0000011

Eg $+6 \longrightarrow 0\ 0000110$

$-9 \longrightarrow 1\ 1110110$

$-3 \qquad\quad 1\ 1111100$

↓ signed to unsigned Conversion (1's comp.)

$-(0000011)_2 = (-3)_{10}$

Eg:- $-9 \longrightarrow 1\ 1110110$

$+(-9) \longrightarrow 1\ 1110110$

$-18 \qquad\quad \boxed{1}\ 1\ 1101100$

$\longrightarrow 1$

$1\ 1101101$

↓ 1's compliment.

$-(0010010)_2 = (-18)_{10}$.

2's compliment Arithmetic

- Carry is ignored here.

Eg:-

$-9 \longrightarrow 1\ 1110111$

$+(-9) \longrightarrow 1\ 1110111$

$-18 \qquad\quad \boxed{1}\ 1\ 1101110$

↓ 2's compliment

$-(0010001 + 1) = -(0010010) = (-18)_{10}$

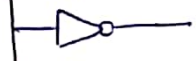
Overflow / Underflow

Overflow : Result greater than largest no. which can be represented (i.e. $\geq 2^n - 1$)

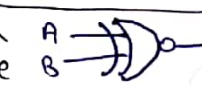
Underflow : Result smaller than smallest no. which is in range. (i.e., $\leq -(2^n)$).

Boolean Algebra & Digital Logic

Basic Gates:

AND		$X = AB$
OR		$X = A + B$
I(NOT)		$X = A'$
Buffer		$X = A$
NAND		$X = (AB)'$
NOR		$X = (A+B)'$

Some special gates:-

XOR (Exclusive OR)		$X = A \oplus B$ $X = A'B + AB'$
XNOR (Exclusive NOR or Equivalence)		$X = (A \oplus B)'$ $X = A'B' + AB$

Boolean Algebra : A Mathematical system for manipulation of variables that can have one of two values. (like 0 & 1, True or False, High or Low).

AND Operator : Boolean product

OR operator : Boolean Sum.

Boolean function has At least :

- one boolean variable
- one boolean operator
- any one of 0 and 1.

★ 0 : Identity element for '+' or (Or) operation. [a + 0 = a]

1 : Identity element for '.' or (AND) operation [a . 1 = a]

* Commutative : $a+b = b+a$

$a \cdot b = b \cdot a$

* Associative : $a+(b+c) = (a+b)+c$.

$a \cdot (b \cdot c) = (a \cdot b) \cdot c$.

* Distributive : $a+(b \cdot c) = (a+b) \cdot (a+c)$

$a \cdot (b+c) = a \cdot b + a \cdot c$.

* Complement : $a+a' = 1$

$a a' = 0$.

* Absorption Property : $a+bc = (a+b)(a+c)$.

Duality

The principal of duality says that if an expression is valid in boolean algebra, dual of that expression is also valid.

* OR gate in +ve logic system becomes ~~and~~ AND gate in -ve logic system.

$F = a+bc$

Then, Dual of $F = \cancel{(a+b)(a+c)}$ $a \cdot (b+c)$. $a \cdot (b+c)$.

* De Morgan's Theorem : $(a+b)' = a'b'$

$(ab)' = a'+b'$

Involution : Taking double Inverse gives same value.

Eg:- $F = (AB'+C)D' + E$. Find F'

$\Rightarrow F' = ((AB'+C)D' + E)'$

$= ((AB'+C)D')' \cdot E'$

$= [(AB'+C)' + D] \cdot E'$

$= [(A'+B) \cdot C' + D] \cdot E'$

Eg:- $F = A + A\bar{B} + \bar{A}B$

$= A(1 + \bar{B}) + \bar{A}B$

$= A + \bar{A}B$

$= (A + \bar{A})(A + B)$

$= \boxed{A + B}$.

Answer.

Boolean Functions

$F = xy + x'z + yz$.

Solution:

$F(x, y, z) = xy + x'z + yz$

$= (xy + x')(xy + z) + yz$

$= (x + x')(y + x')(xy + z) + yz$

$= (x' + y)(xy + z) + yz$.

$= x'z + xy + yz(x + x')$.

$= xy + x'z$.

x \ yz	00	01	11	10
0		1	1	
1			1	1

SOP Form.

( ) + ( ) + ( )

↓ min term (m)

* Standard gate for SOP is NAND gate & for POS is NOR gate.

Eg:-

x	y	z	minterms	
0	0	0	$x'y'z'$	m_0
0	0	1	$x'y'z$	m_1

1	0	0	$xy'z'$	m_4

1	1	1	xyz	m_7

Eg :-

	x	y	F
$m_0 \rightarrow$	0	0	0
$m_1 \rightarrow$	0	1	1
$m_2 \rightarrow$	1	0	0
$m_3 \rightarrow$	1	1	0

$F = m_1 = x'y \cdot z \quad \Sigma m(1)$

Eg :- $F = m_1 + m_3 + m_5 = \Sigma m(1,3,5)$.

Solution: 001 011 101

$F = x'y'z + x'yz + xy'z$

$\Rightarrow F = x'z(y'+y) + xy'z$

$\Rightarrow F = x'z + xy'z$

$\Rightarrow F = (x'+xy')z = (x'+x)(x'+y')z$

$= \boxed{x'z + y'z}$.

Implicant

★ Every term in SOP is called an Implicant (i.e. for which output is 1).

★ Prime Implicant : ~~A * prime~~ from which no literals can be removed.

Eg : $F = \Sigma(2,6,7)$.

Solution :-

$F = a'bc' + abc' + abc$

Implicants

$\Rightarrow F = \boxed{a'bc' + ab}$

Prime Implicants

$\Rightarrow F = b(a'c' + a) = b(a'+a).(c'+a) = bc' + ab$.

Eg :- $F(a,b) = \Sigma\, m(1,2)$.

Solution :-

$F(a,b) = a'b + ab'$

(Implicants / Min terms / Prime Implicants)
[∵ They can not be reduced further]

Eg : $F = A + B'C$. Express in SOP.

Solve :-

$F = A + B'C$

$\Rightarrow F = A(B+B')(C+C') + (A+A')B'C$

$\Rightarrow F = ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$.

$\Rightarrow F = ABC + ABC' + AB'C + AB'C' + A'B'C$

$\Rightarrow \boxed{F = \Sigma m(7,6,5,4,1)}$

$\Rightarrow F = \boxed{\Sigma m(1,4,5,6,7)}$ Answer.

Eg :- $F = AC' + A'C$. Express in SOP.

Solution :

$F = A(B+B')C' + A'(B+B')C$

$= ABC' + AB'C' + A'BC + A'B'C$

$\Rightarrow F = \Sigma m(6,4,3,1)$

$\Rightarrow \boxed{F = \Sigma m(1,3,4,6)}$ $\Rightarrow F(A,B,C) = \boxed{\Sigma m(1,3,4,6)}$ Answer.

Eg :- $F(A,B,C,D) = AC$.

Solution :

$F = A(B+B')C(D+D')$

$\Rightarrow F = ABCD + ABCD' + AB'CD + AB'CD'$

$\Rightarrow F = \Sigma m(15,14,11,10)$

$\Rightarrow F = \Sigma m(10,11,14,15)$

POS Form
(Product of Sum).

$(\ +\ +\ ).(\ +\ +\ ).(\ +\ +\ )$

↑ Max terms

x	y	z	Max term	
0	0	0	$x+y+z$	M_0
0	0	1	$x+y+z'$	M_1
...				
1	0	0	$x'+y+z$	M_4
...				
1	1	1	$x'+y'+z$	M_7

Eg :-

x	y	z	F	
0	0	0	0	M_0
0	0	1	0	M_1
0	1	0	1	M_2
0	1	1	0	M_3
1	0	0	0	M_4
1	0	1	1	M_5
1	1	0	1	M_6
1	1	1	1	M_7

$F = (x+y+z)(x+y+z').(x+y'+z').(x'+y+z).$

$= \prod m(0,1,3,4).$

Eg :- $F(A,B,C,D) = (A+B'+C)(A+C'+D).$

Solution :-

$F(A,B,C,D) = (A+B'+C+DD')(A+BB'+C'+D)$

$= (A+B'+C+D)(A+B'+C+D')(A+B+C'+D)$

$\quad\quad (A+B'+C'+D)$

$=$

Eg:- $F(a,b,c) = \Sigma m(0,2,4,6)$. We can Interchange them easily.

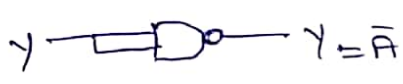
$\Rightarrow F(a,b,c) = \Pi M(1,3,5,7)$.

Universal Logic Gates

- NAND
- NOR

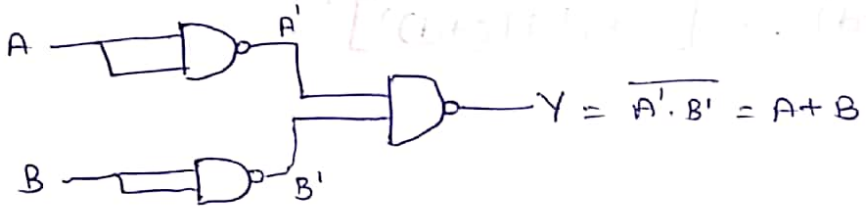
→ Any basic gate can be implemented using any of the above two gates.

*NAND Gates into other gates.



NOT Gate

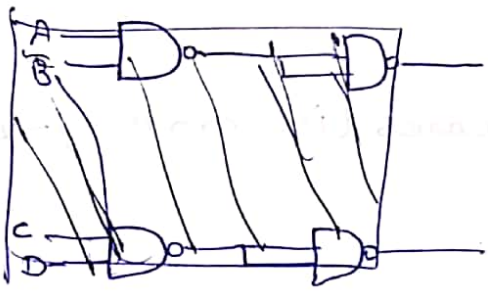
AND gate

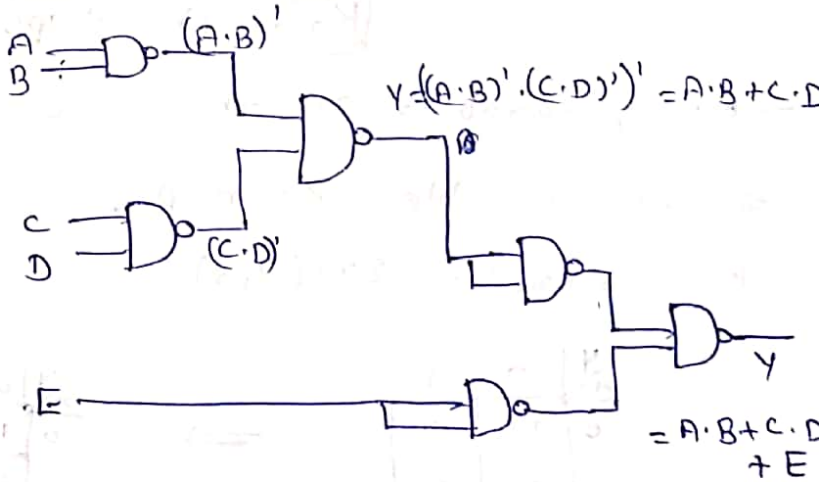


~~*NOR Gate~~ OR gate

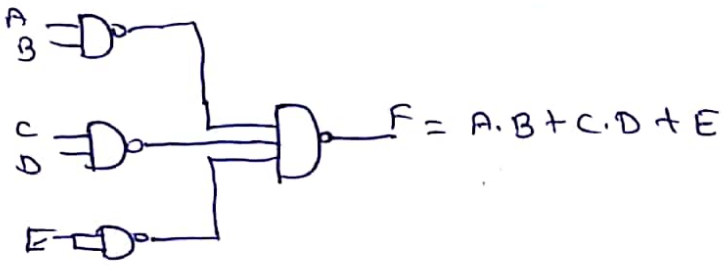
Eg:- $f = AB + CD + E$. Implement using only NAND gates

Solve:

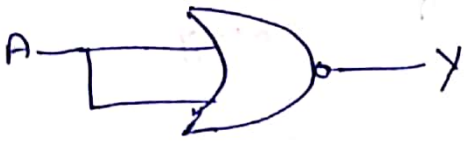




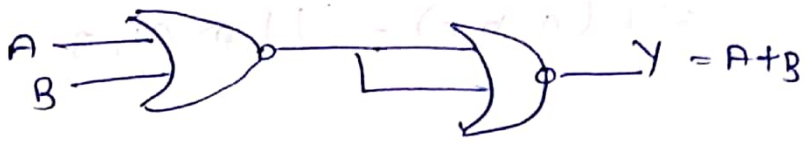
(Or)



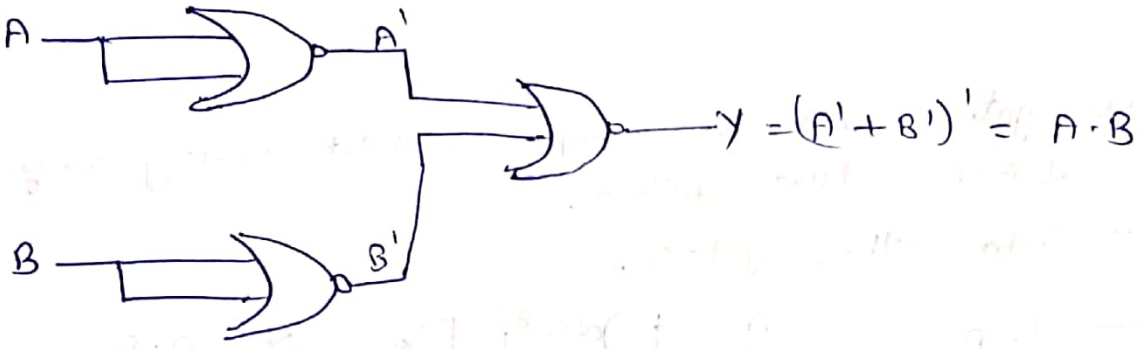
* NOR Gates into other gates.



NOT gate

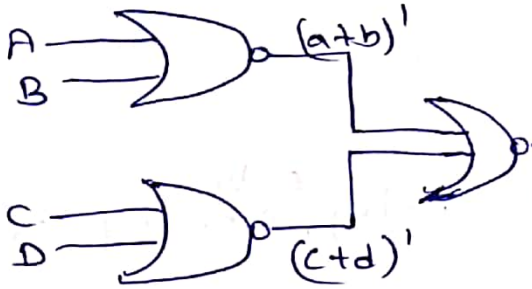


OR gate



AND GATE

Eg: $F = (a+b)(c+d) = [(a+b)' + (c+d)']'$

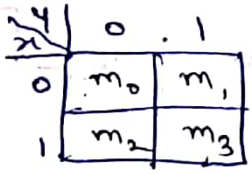


Minimisation with Karnaugh Maps

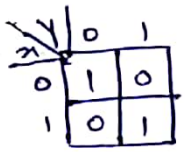
→ No. of variables = n, then no. of squares (i.e. minterms) = 2^n.

Two variable K-map

Eg:- $F(x,y) = \Sigma m(0,3)$

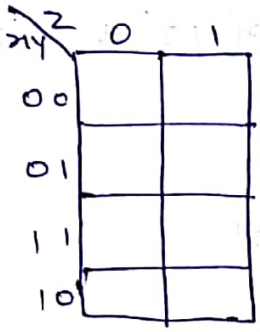


⟹

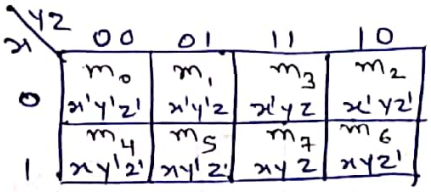


⇒ $F = x'y' + xy$

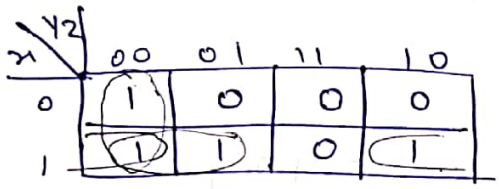
3-variable K-map :



(or)

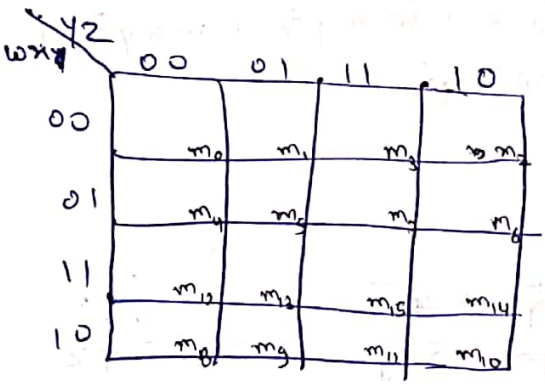


Eg :- $F(x,y,z) = \Sigma m(0,4,5,6)$.



$F = y'z' + xy' + xz'$

4-variable K-map :

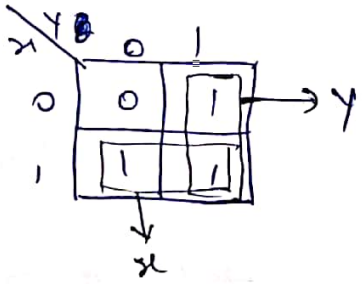


Simplification using K-map.

Pair - a group of 2 adjacent squares
Quad - a group of 4 adjacent squares
Octet - a grout of 8 adjacent squares

[1st preference will be to higher order groups]

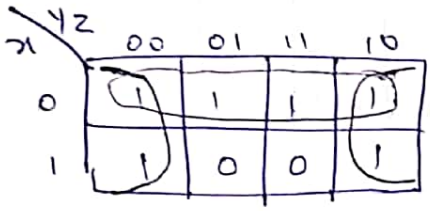
Eg :- $F(x,y) = \Sigma m(1,2,3)$



$\Rightarrow \boxed{F = x+y}$

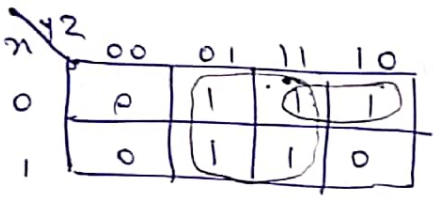
Eg :-

Eg :- $F(x,y,z) = x'y'z' + x'y'z + x'yz + xy'z' + xy'z + xyz'$



$F(x,y,z) = x' + z'$.

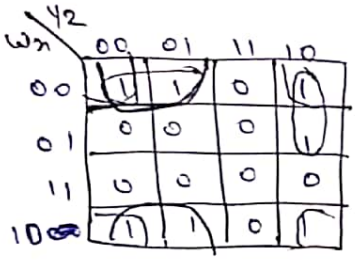
Eg :- $F(x,y,z) = \Sigma m(1,2,3,5,7)$



$\Rightarrow F = z + x'y$.

Eg :- $F(w,x,y,z) = \bar{w}\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}\bar{y}z + \bar{w}\bar{x}y\bar{z} + \bar{w}x\bar{y}\bar{z} + \bar{w}x\bar{y}z + \bar{w}xy\bar{z}$

Solve :-



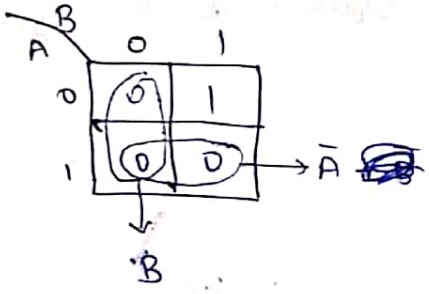
$F = \bar{x}\bar{y} + \bar{z}\bar{x} + \bar{w}y\bar{z}$

Simplification of POS expression using K-map.

2 variable :-

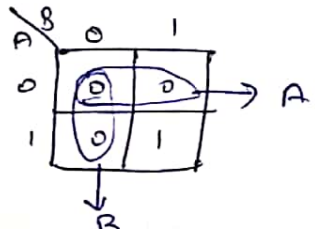
Eg :-

(1)



$\Rightarrow F = \bar{A} \cdot B$

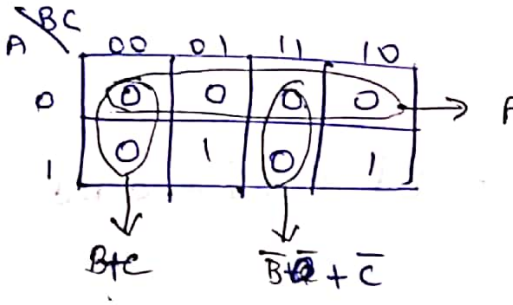
(2)



$\Rightarrow F = A \cdot B$.

3 Variable Expression :-

$F(A,B,C) = \prod (0,1,2,3,4,7)$.



$$F = A \cdot (B+C)(\bar{B}+\bar{C})$$

~~4 Variable Expression :-~~

Don't Care Conditions

→ Invalid set of inputs which give unspecified output.

→ Particular set of inputs which can never happen.

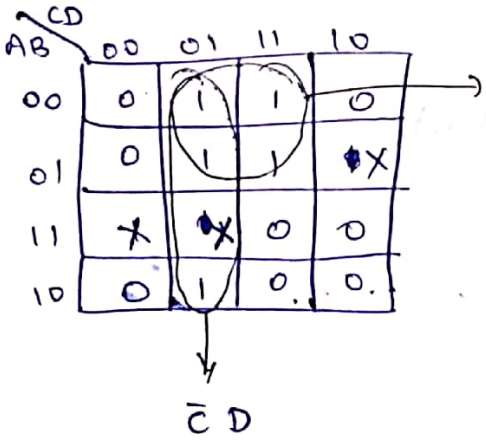
Eg:- In BCD, 0-9 are valid

But, 10-15 are invalid. So, They come under Don't care condition.

K-maps : Don't cares.

Eg:- $f(A,B,C,D) = \Sigma m(1,3,5,7,9) + d(6,12,13)$

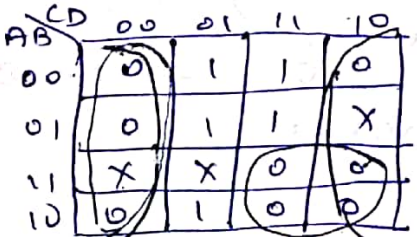
Solution :-



$\Rightarrow F = \bar{A}D + \bar{C}D$

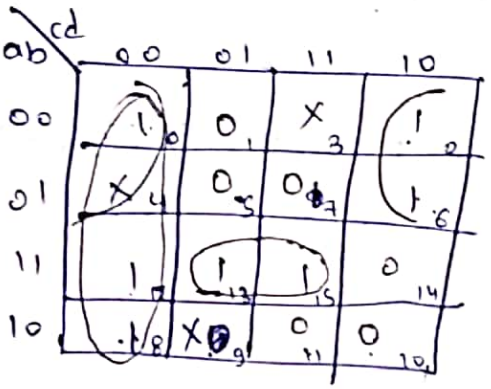
Same Expression.

Eg:- $f(A,B,C,D) = \prod m(0,2,4,8,10,11,14,15) + d(6,12,13)$



$F = D \cdot (\bar{A}+\bar{C})$.

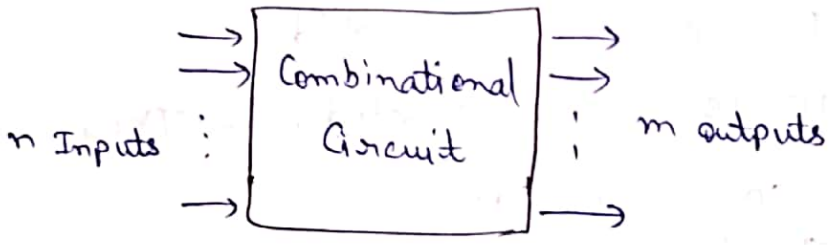
Eg :- $F(a,b,c,d) = \Sigma m(0,2,6,8,12,13,15) + d(3,4,9)$



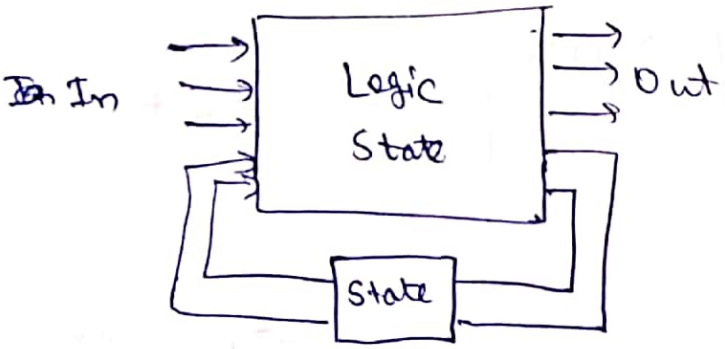
$F = \bar{c}\bar{d} + \bar{a}\bar{d} + a\,b'd$

Combinational Circuits

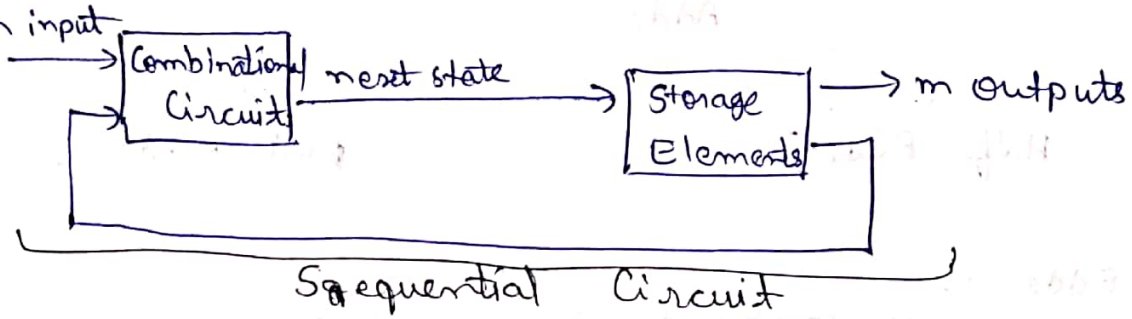
Logic circuits for a digital circuit may be combinational or sequential.



Sequential circuit



★ Combinational circuits are memory-less while Sequential circuit consists of combinational circuit & memory elements.



Design Combination Circuit

Steps

(1) Determine no. of inputs & outputs.

(2) Assign name to input & outputs variables.

(3) Derive Truth table.

(4) Obtain simplified Boolean expression.

(5) Draw Logic Diagram & Verify correctness.

Notes:

- If there are n input variable, then there are 2^n input combinations.
- Each input combination has one output value.
- Truth tables have all combinations of input & corresponding outputs.

Examples:

→ Adder / Subtractor

→ Code converters.

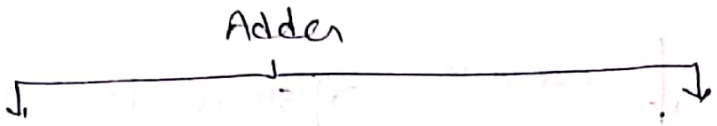
→ Comparators

→ Decoders

→ Encoders

→ Multiplexers.

→ Demultiplexer.



Half Adder Full Adder.

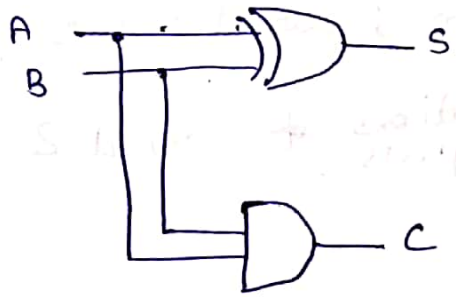
(1) Half Adder :- A combinational circuit that adds 2 inputs bits to generate a Sum bit & a Carry bit.

A	B	Sum S	Carry C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum = $\bar{A}B + A\bar{B}$ = $A \oplus B$

Carry = $A \cdot B$.

Logic Diagram :-



Block Diagram :-

---

(2) Full Adder :

Adds 3 input bits to generate a Sum bit & a Carry bit.

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

By K - map :

Sum S :

x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$S = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$

$\Rightarrow S = \bar{x}(\bar{y}z + y\bar{z}) + x(\bar{y}\bar{z} + yz)$

$\Rightarrow S = \bar{x}(y \oplus z) + x(y \oplus z)'$

$\Rightarrow \boxed{S = x \oplus y \oplus z}$ [Let $y \oplus z = A$ & solve]

Similarly :-

Carry C :

x \ yz	00	01	11	10
0	0	0	1	0
1	0	1	1	1

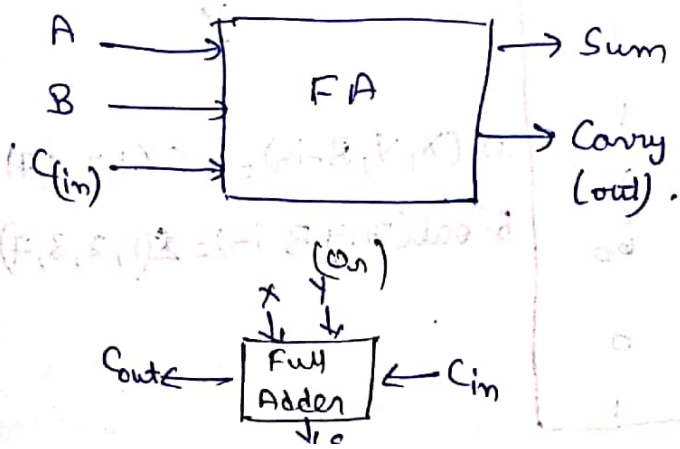
$\Rightarrow \boxed{C = yz + xy + xz}$

$C = \bar{x}yz + x\bar{y}z + xyz + xy\bar{z}$

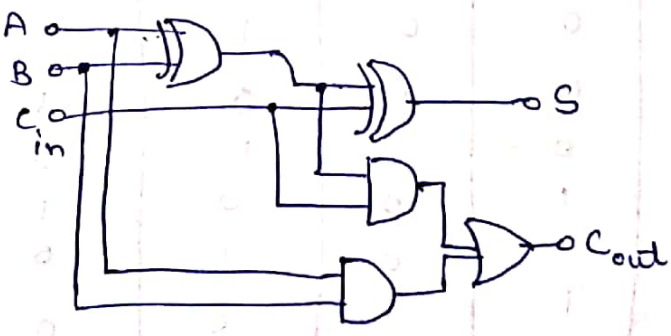
$\Rightarrow C = (\bar{x}y + x\bar{y})z + xy$

$\Rightarrow \boxed{C = (x \oplus y)z + xy}$ → (Alternate Expression).

Block Diagram :-



Logic Diagram :-



→ Subtractor ←

- Subtractor is of two types.
  - → Half Subtractor
  - → Full Subtractor.

Half Subtractor :-

- Subtracting a single-bit binary value Y from another X produces a difference bit D & a borrow out B.

X	Y	D	Bout
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

So, $D(X,Y) = \Sigma(1,2)$

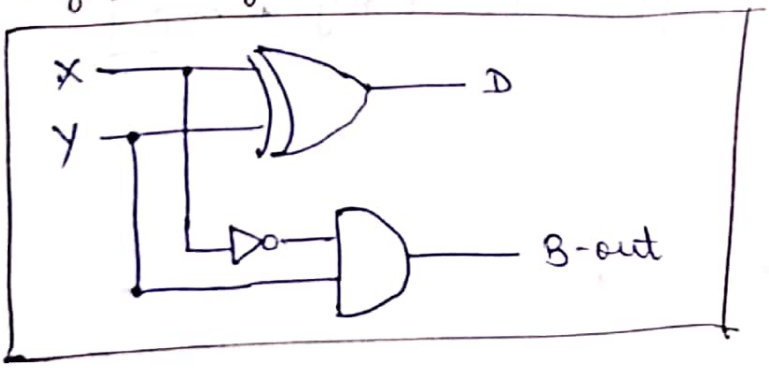
$\Rightarrow D = X'Y + XY'$

$\Rightarrow \boxed{D = X \oplus Y}$.

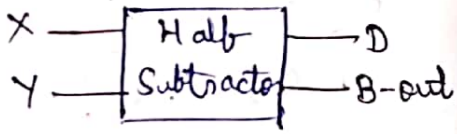
$B\text{-out}(X,Y) = \Sigma(1)$:

$\Rightarrow \boxed{B\text{-out} = X'Y}$

Logic Diagram

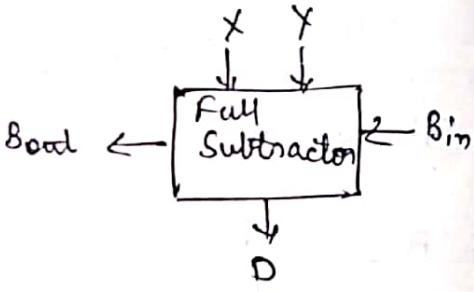


Block-Diagram :-



Full-Subtractor :-

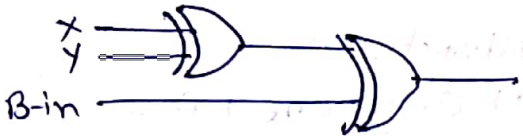
X	Y	B-in	D	B-out
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

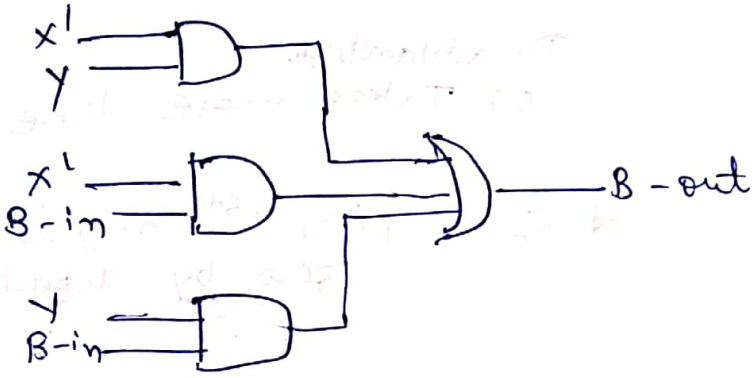


$D(X,Y,B\text{-in}) = \Sigma(1,2,4,7)$

$B\text{-out}(x,y,B\text{-in}) = \Sigma(1,2,3,7)$

Logic - Diagram :-





- 4-Bit Adders :-
  - Inputs - 9 = (2x4+1)
  - Outputs - 5 = (4+1)
  - Size of Truth-table - 512
  - How many functions to optimize : 5 functions.

Eg :-

Ai		A_3	A_2	A_1	A_0
Bi		B_3	B_2	B_1	B_0
					C_0

Total 9 Inputs

Bout	D_3	D_2	D_1	D_0

5 outputs

* Binary Addition can be done in two ways :-

Serial

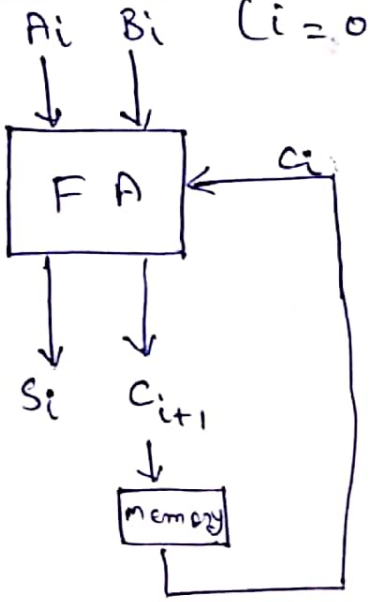
uses only one FA circuit & a storage device, to store the generated output carry.

Parallel

uses n FAs & all bits of the two numbers along with the input carry can be applied simultaneously.

→ Serial Addition :-

4-bit serial Adder.



Advantage :-

(1) Only one FA is required.

(2) less memory is required.

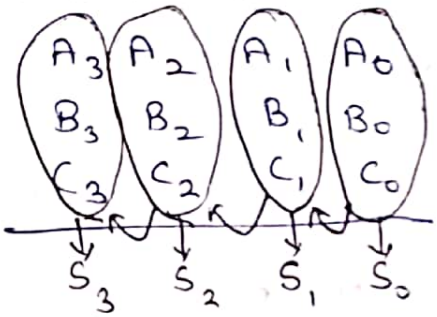
Disadvantage :-

(1) Takes more time.

* $C_0 = 0$ [i.e, 1st Carry is always zero by default]

~~Advantage~~ :

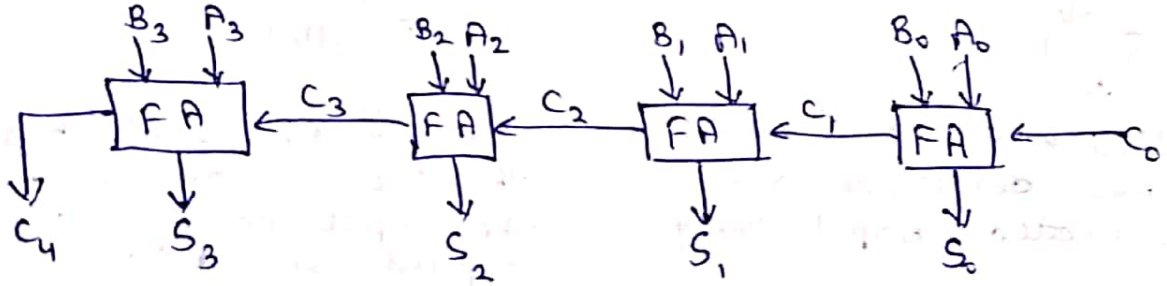
→ Parallel Addition :-



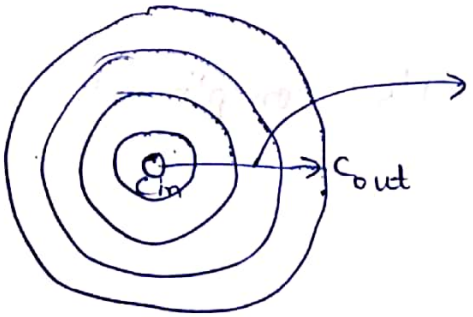
Eg :-

```
  0  1  0  1
  0  1  1  0
  1  0  0  0   → Carry.
  1  0  1  1
```

* 4-bit binary parallel Adder :-

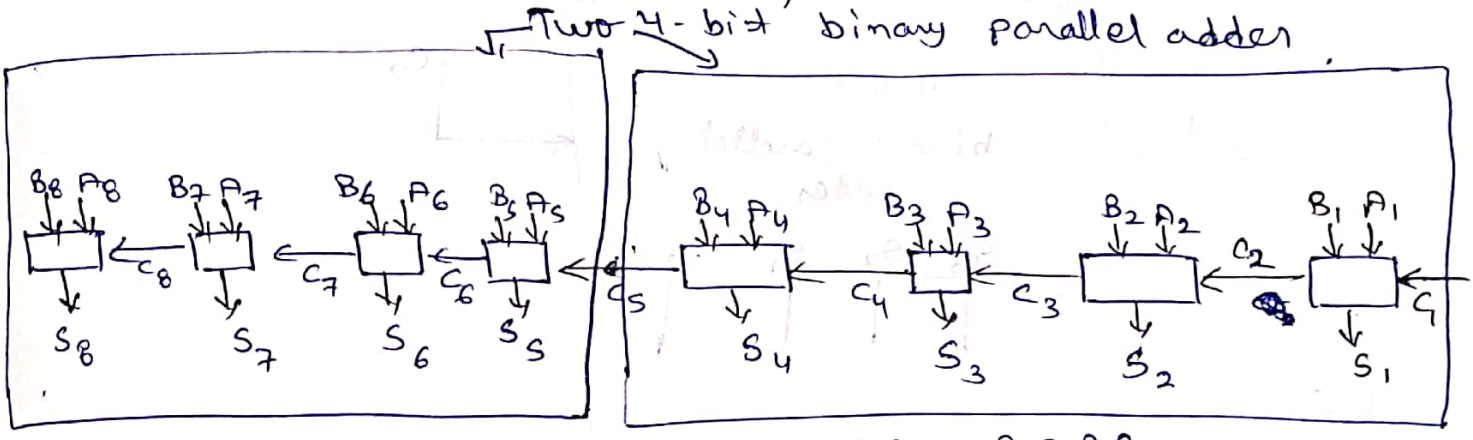


This adder is also called ripple carry adder.

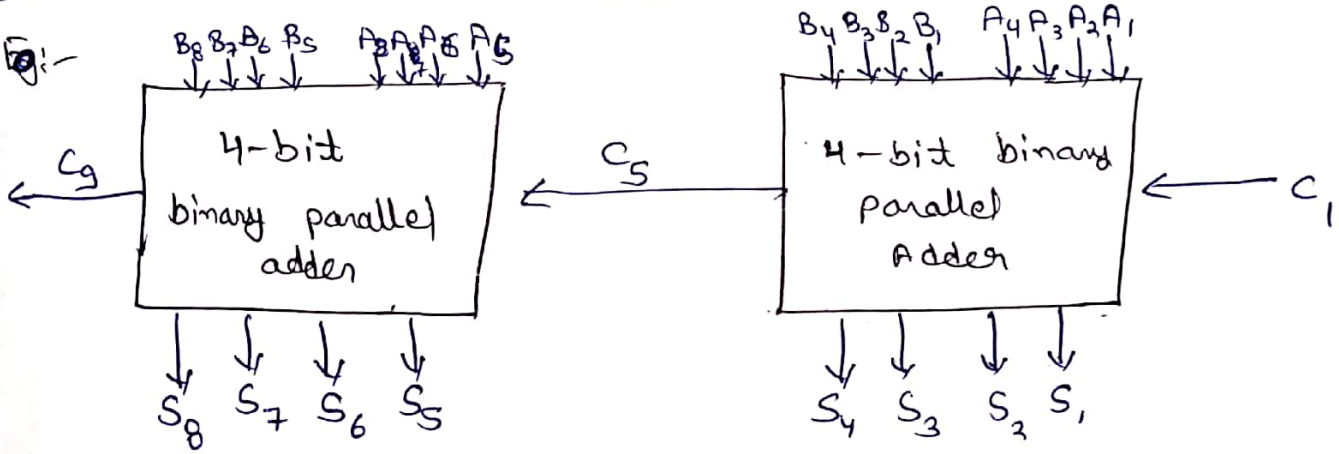


→ Ripple Propagation Delay.

Cascading of Parallel adders
(See slides)



Eg:-



Ripple-Adder Delay:-

- Delay = $(2n+1)T$

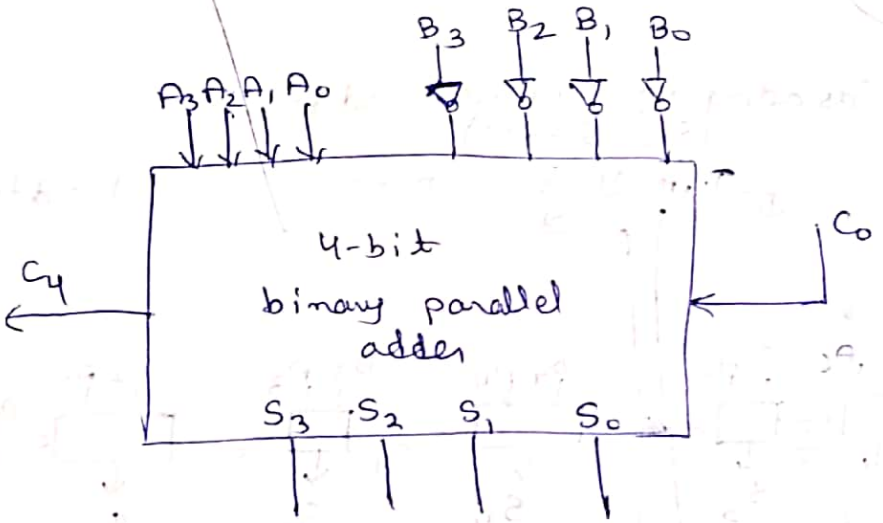
Carry-Look Ahead Adder

→ Generate all incoming carries in parallel.

→ Based on two addition functions of FA, called Carry Generate (G_i) & Carry propogate (P_i).

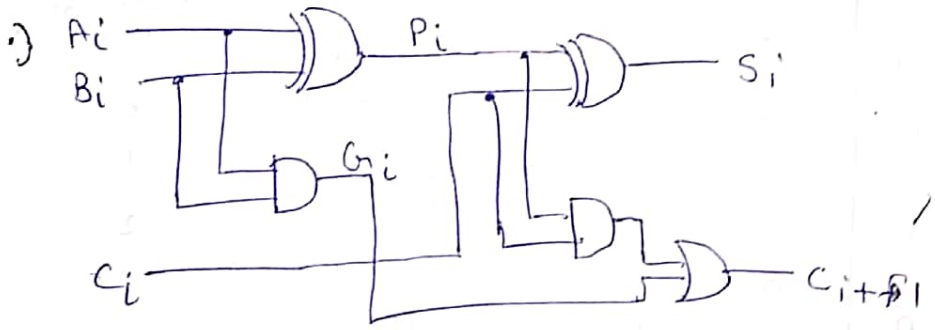
For A - B operation using 1's compliment:-

4-bit binary parallel adder:-



(Or)

Carry – Look Ahead Adder :–



$P_i = A_i \oplus B_i$, $G_i = A_i B_i$

$S_i = P_i \oplus C_i$; $C_{i+1} = G_i + P_i C_i$

Thus,

•) $C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0)$

$= G_1 + P_1 G_0 + P_1 P_0 C_0$.

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0)$

$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

•) $C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 +$

$P_3 P_2 P_1 P_0 C_0$.

→ P_i is carry propagate & G_i is carry generate

→ Since C_4 is independent of C_3, C_2 & C_1.

So, all carries can be generated in parallel without waiting for previous carries to be generated.

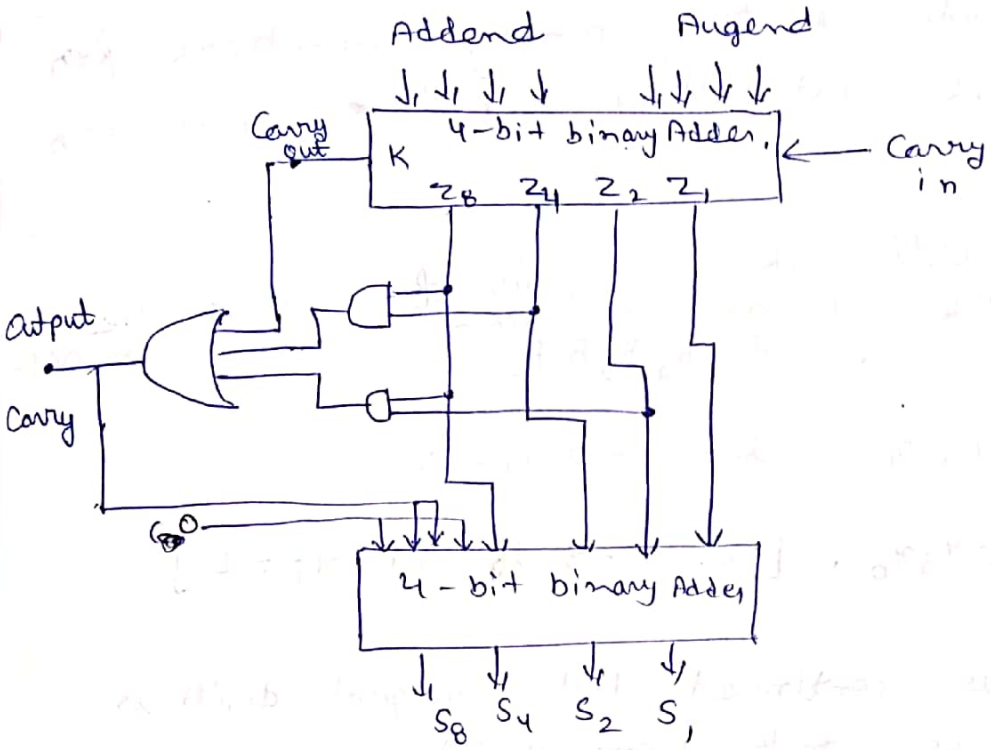
	Carry-look ahead Adder	Ripple Adder
* Time – Delay :	4T	(2n+1)T.

BCD Adder :-

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

* $C_{(output)} = K_{(16-19)} + Z_8 Z_4 + Z_8 Z_2$
$(12-15)$ $(10-11)$.

Block - diagram :-



H.W :- Design a combinational circuit which produces square of a given input.

Solution :-

	I_2	I_1	I_0	$S_{32}S_5$	$S_{16}S_4$	S_8S_3	S_4S_2	S_2S_1	S_1S_0
0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1
2	0	1	0	0	0	0	1	0	0
3	0	1	1	0	0	1	0	0	1
4	1	0	0	0	1	0	0	0	0
5	1	0	1	0	1	1	0	0	1
6	1	1	0	1	0	0	1	0	0
7	1	1	1	1	1	0	0	0	1

S_0, $S_5 = I_2 I_1$, $S_4 = I_2 I_1' + I_2 I_1 I_0$

$S_3 = I_2' I_1 I_0 + I_2 I_1' I_0$

$S_2 = I_1 I_0'$

$S_1 = 0$

$S_0 = I_0$

$\rightarrow S_3 = (I_2 \oplus I_1) I_0$

#Magnitude Comparator

- Compares magnitude of two binary numbers for the purpose of establishing whether $a > b$, $a = b$ or $a < b$.

- The equality relation of each pair of bits can be expressed logically with an exclusive-NOR function as:

$A = A_3 A_2 A_1 A_0$; $B = B_3 B_2 B_1 B_0$

$x_i = A_i B_i + A_i' B_i'$ for $i = 0, 1, 2, 3$

$(A = B) = x_3 \cdot x_2 \cdot x_1 x_0$. [i.e., $A = B$ if all $x_i = 1$]

- The process is continued till unequal digits is reached (Process starts from MSB)
If corresponding digit of A is 1 & of B is 0, we conclude that $A > B$.

$A > B = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$

$A < B = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$.

Eg: (1) $A = 1\ 0\ 0\ 1\ 0$
$B = 1\ 0\ 0\ 0\ 1$
$\Rightarrow A_4 = B_4$, $A_3 = B_3$, $A_2 = B_2$, $A_1 > B_1 \Rightarrow A > B$.

Here, $A_4 B_4' + x_4 A_3 B_3' + x_4 x_3 A_2 B_2' + x_4 x_3 x_2 A_1 B_1' + x_4 x_3 x_2 x_1 A_0 B_0' = 0 + 0 + 0 + 1 + 0 = 1$

(2) $A = 1\ 0\ 1\ 0$
$B = 1\ 1\ 0\ 1$
$\Rightarrow A_3 = B_3$, $A_2 < B_2 \Rightarrow A < B$.

Here, $A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0 = 0 + 1 = 1 \Rightarrow A < B$

(3) $A = 1\ 0\ 1\ 1$
$B = 1\ 0\ 1\ 1$
$\Rightarrow A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, $A_0 = B_0 \Rightarrow A = B$.

Here, $x_3 = x_2 = x_1 = x_0 = 1 \Rightarrow A = B$.

Read 4-bit mag. Comparator circuit from slides.

Code Converters

- A logic circuit that changes data presented in one type of binary to another type of binary code

(a) BCD to Excess-3 code converters :-

- (10-15) will be treated as "don't cares."

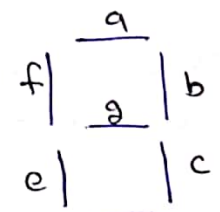
BCD					XS3		
A	B	C	D	W	X	Y	Z
0	0	0	0 →	0	0	1	1
0	0	0	1 →	0	1	0	0
0	0	1	0 →				
0	0	1	1 →				
0	1	0	0 →				
0	1	0	1 →				
0	1	1	0 →				
0	1	1	1 →				
1	0	0	0 →				
1	0	0	1 →	1	1	0	0
1	0	1	0 →	x	x	x	x
1	0	1	1 →	x	x	x	x
1	1	0	0 →	x	x	x	x
1	1	0	1 →	x	x	x	x
1	1	1	0 →	x	x	x	x
1	1	1	1 →	x	x	x	x

On Solving by K-map :-

$Z = D'$, $Y = CD + C'D'$

(b) BCD to Seven-Segment :-

- Each segment is named a, b, c, d, e, f, g.



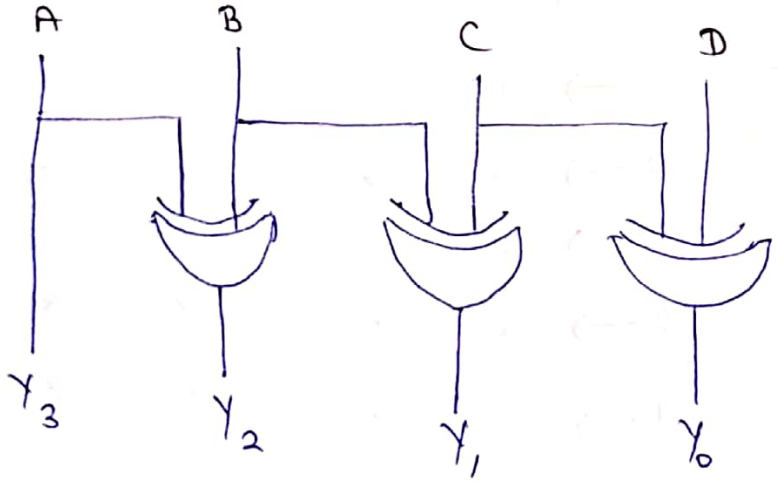
Seven-Segment Display

- Eg :-

	BCD				Seven-Segment						
	w	x	y	z	a	b	c	d	e	f	g
(0) →	0	0	0	0	1	1	1	1	1	1	0
(1) →	0	0	0	1	0	1	1	0	0	0	0
(2) →	0	0	1	0	1	1	0	1	1	0	1
(3) →	0	0	1	1	1	1	1	1	0	0	1
(4) →	0	1	0	0	0	1	1	0	0	1	1
(5) →	0	1	0	1	1	0	1	1	0	1	1
(6) →	0	1	1	0	1	0	1	1	1	1	1
(7) →	0	1	1	1	1	1	1	0	0	0	0
(8) →	1	0	0	0	1	1	1	1	1	1	1
(9) →	1	0	0	1	1	1	1	1	0	1	1

(c) Binary to GRAY : Do yourself.

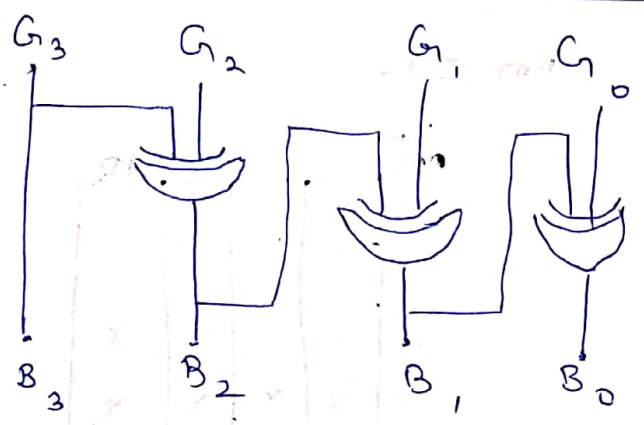
$Y_3 = A$, $Y_2 = A \oplus B$, $Y_1 = B \oplus C$ & $Y_0 = C \oplus D$.



(d) GRAY to Binary :

$B_3 = G_3$, $B_2 = G_2 \oplus G_3$, $B_1 = G_1 \oplus G_2 \oplus G_3$,

$B_0 = G_0 \oplus G_1 \oplus G_2 \oplus G_3$.



(e) GRAY to BCD converter : (Assignment) ★

★ Assignment : Design a 3-bit parity generator & checker.

GRAY Code				BCD CODE			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	X	X	X	X
1	1	1	0	X	X	X	X
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	0	0	1	X	X	X	X
1	0	0	0	X	X	X	X

K-Map :

For W :-

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	X	X
10	X	X	X	X

$W = A$

For X :-

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	X	X
10	X	X	X	X

$\bar{A}B$

$X = \bar{A}B$

For Y :-

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	X	X
10	X	X	X	X

$Y = \bar{B}C + \bar{A}B\bar{C}$

$= (A+\bar{A})\bar{B}C + \bar{A}B\bar{C}$

$= A\bar{B}C + \bar{A}(B \oplus C)$

For Z :-

AB \ CD	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	X	X
10	X	X	X	X

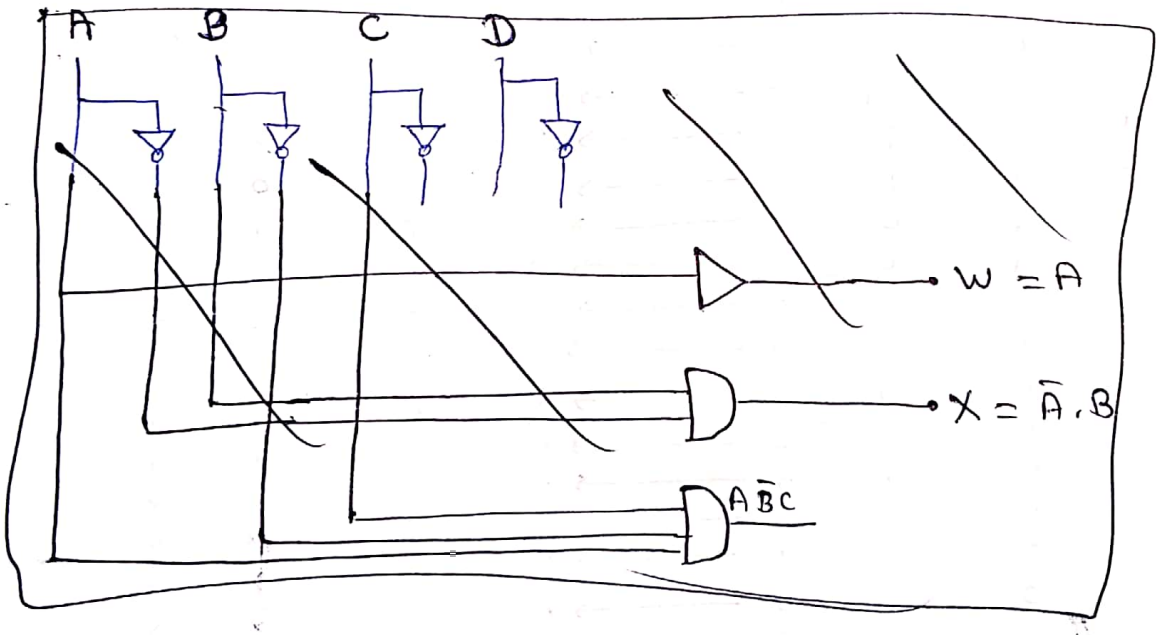
$Z = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BCD$

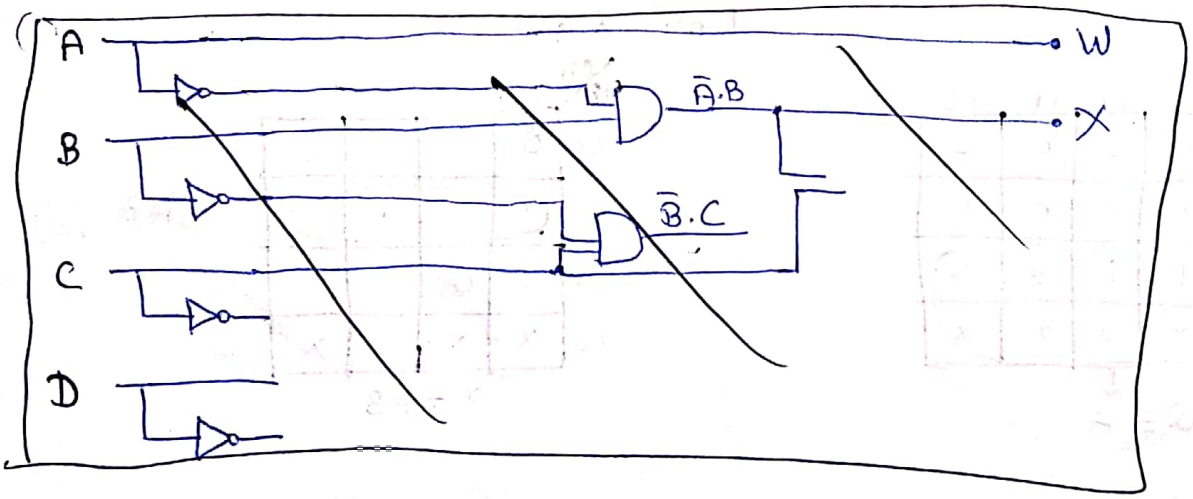
$= \bar{A}\bar{B}(C \oplus D) + \bar{A}B\overline{(C \oplus D)}$

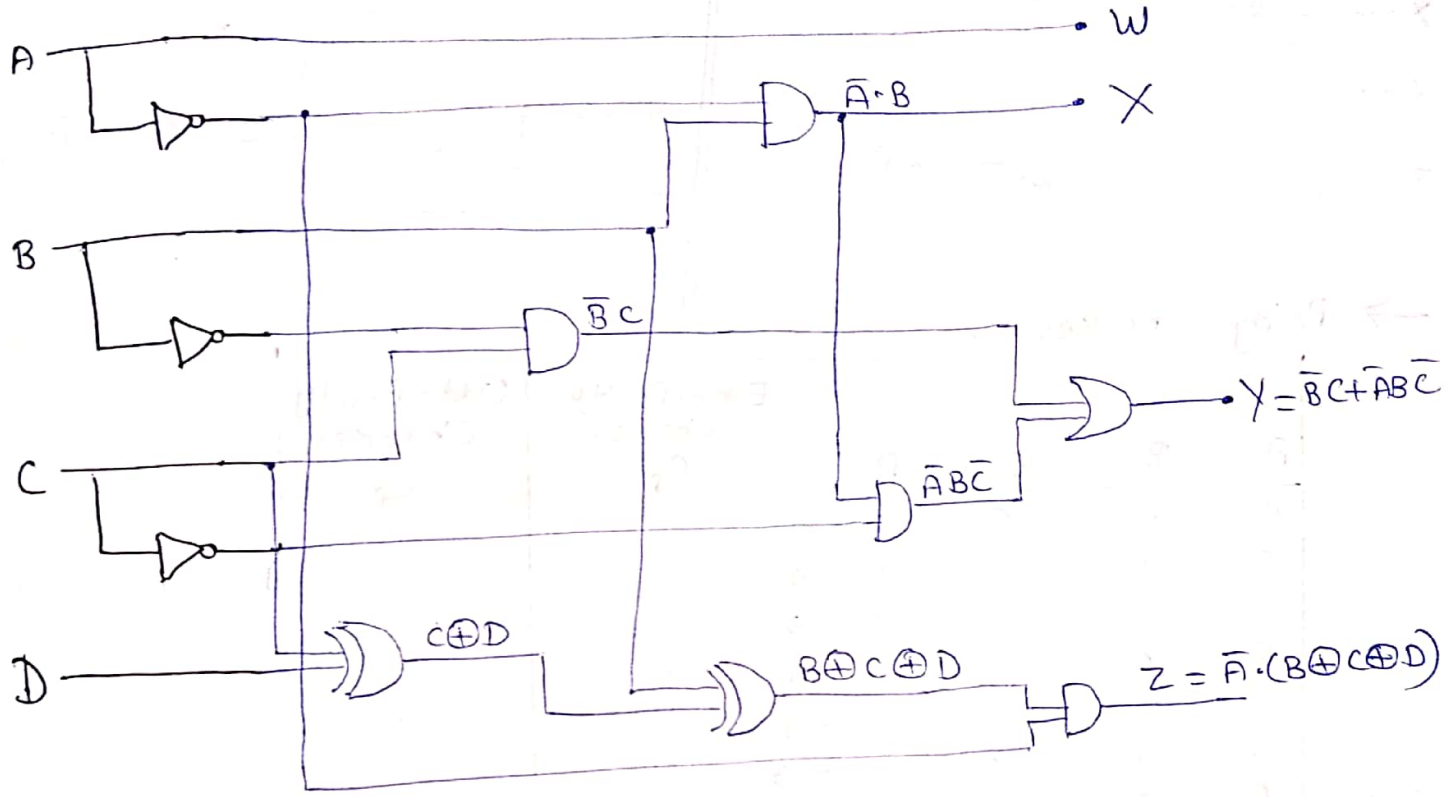
$= \bar{A}[\bar{B}(C \oplus D) + B\overline{(C \oplus D)}]$

$= \bar{A}.[B \oplus C \oplus D]$

Circuit Diagram :







ASSIGNMENT :-

3-bit parity Generator & Checker,

→ For Parity Generator :-

3-bit message			Odd-Parity Bit(P)	Even-Parity bit(P)
X	Y	Z		
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

K-Map Simplification :-

Even - Parity

X \ YZ	00	01	11	10
0	0	(1)	0	(1)
1	(1)	0	(1)	0

$P = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$

$= \bar{X}(Y \oplus Z) + X(\overline{Y \oplus Z})$

$= \boxed{X \oplus Y \oplus Z}$

Odd - Parity

X \ YZ	00	01	11	10
0	(1)	0	(1)	0
1	0	(1)	0	(1)

$P = \bar{X}\bar{Y}\bar{Z} + \bar{X}YZ + X\bar{Y}Z + XY\bar{Z}$

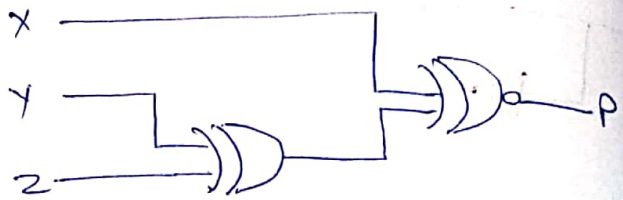
$= \bar{X}(\overline{Y \oplus Z}) + X(Y \oplus Z)$

$= \boxed{\bar{X} \oplus Y \oplus Z}$

CIRCUIT :-

Even - Parity

Odd - Parity.



→ Parity Checker :-

				Even-Parity Checker	Odd-Parity Checker
A	B	C	P	C_p	C_p
0	0	0	0 →	0	1
0	0	0	1 →	1	0
0	0	1	0 →	1	0
0	0	1	1 →	0	1
0	1	0	0 →	1	0
0	1	0	1 →	0	1
0	1	1	0 →	0	1
0	1	1	1 →	1	0
1	0	0	0 →	1	0
1	0	0	1 →	0	1
1	0	1	0 →	0	1
1	0	1	1 →	1	0
1	1	0	0 →	0	1
1	1	0	1 →	1	0
1	1	1	0 →	1	0
1	1	1	1 →	0	1

Simplification By K-Map :

Even-Parity checker

AB \ CP	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$C_p = \bar{A}\bar{B}\bar{C}P + \bar{A}\bar{B}C\bar{P} + \bar{A}B\bar{C}\bar{P} + \bar{A}BCP + AB\bar{C}P + AB\bar{C}\bar{P} + A\bar{B}\bar{C}\bar{P} + A\bar{B}CP$

$= \bar{A}\bar{B}(C \oplus P) + \bar{A}B(\overline{C \oplus P}) + AB(C \oplus P) + A\bar{B}(\overline{C \oplus P})$

$= \bar{A}[C \oplus P \oplus B) + A(B \oplus C \oplus P)$

$= [A \oplus B \oplus C \oplus P]$

Odd - Parity Checker.

AB \ CP	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

$C_p = \bar{A}\bar{B}\bar{C}\bar{P} + \bar{A}\bar{B}CP + \bar{A}B\bar{C}P + \bar{A}BC\bar{P} + AB\bar{C}\bar{P} + ABCP + A\bar{B}\bar{C}P + A\bar{B}C\bar{P}$

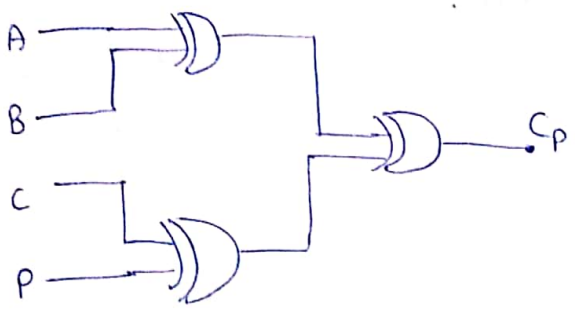
$= \bar{A}[\bar{B}(C \oplus P) + B(\overline{C \oplus P})] + A[B(C \oplus P) + \bar{B}(\overline{C \oplus P})]$

$= (A \oplus B)(\overline{C \oplus P}) + (\overline{A \oplus B})(C \oplus P)$

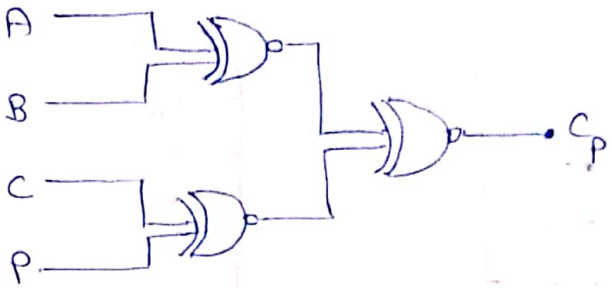
$= ((A \oplus B) \oplus (C \oplus P))'$

Circuit Diagram:-

Even-Parity Checker



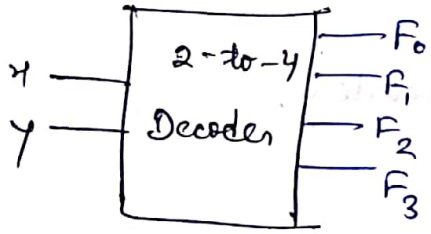
Odd-Parity Checker.



Decoder

- Digital circuit that converts an input binary code into a single numeric output.
- n inputs &
  2^n outputs. (Max. outputs; may be less than 2^n if there are don't cares)
- n to 2^n - line decoder.
- Generates all the minterms of n input variables,
- Exactly one output will be active for each combination of outputs.

2-to-4 Binary Decoder.

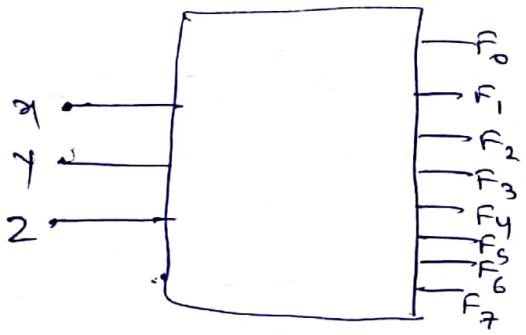


X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

★Each output is a 2 variable min-term ($\bar{X}\bar{Y}, \bar{X}Y, X\bar{Y}, XY$).

3-to-8 Binary Decoder:

Also called (Binary to octal converter).



4-to-10 line decoder.

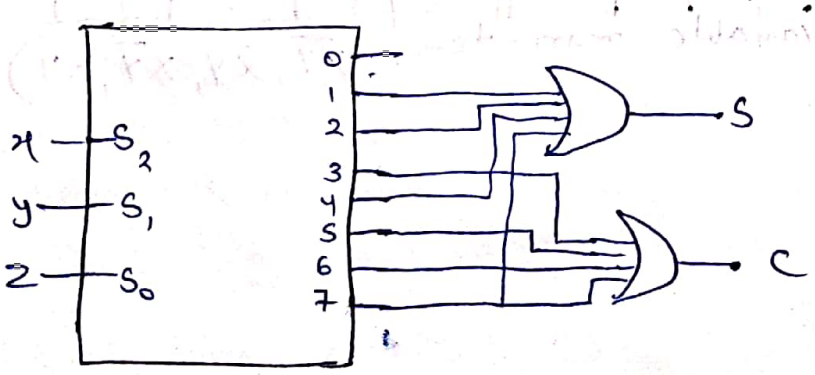
Eg:- BCD to Decimal

w	x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1

Implementing Functions using Decoder.

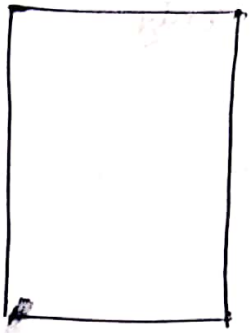
Eg:- $S(x,y,z) = \Sigma(1,2,4,7)$

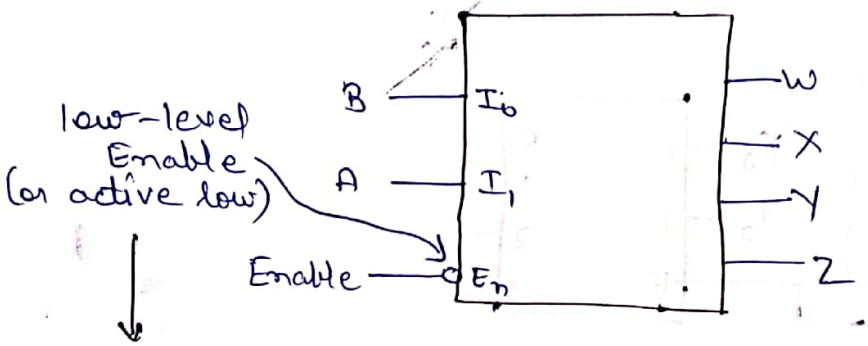
$C(x,y,z) = \Sigma(3,5,6,7)$



★ Decoder with Enable :-

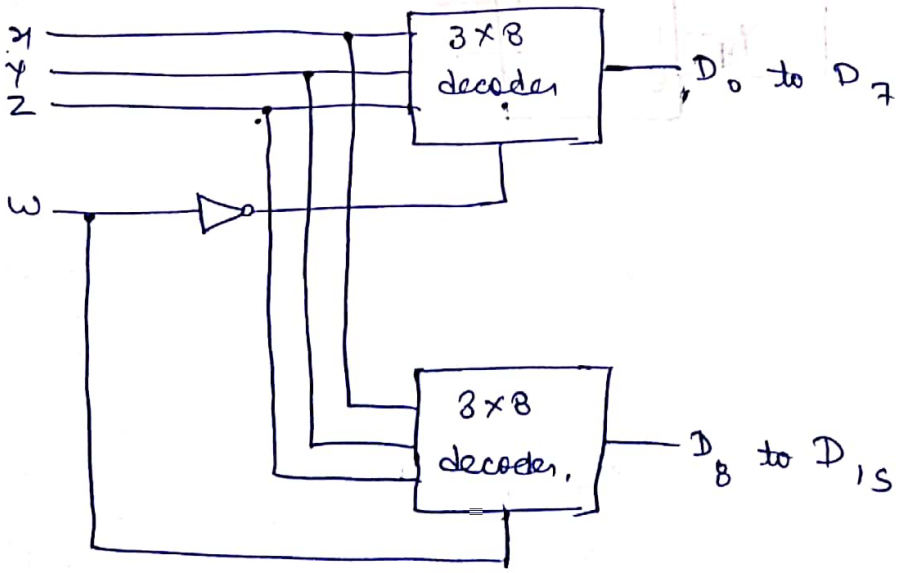
E_0	A	B	W	X	Y	Z
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	X	X	0	0	0	0



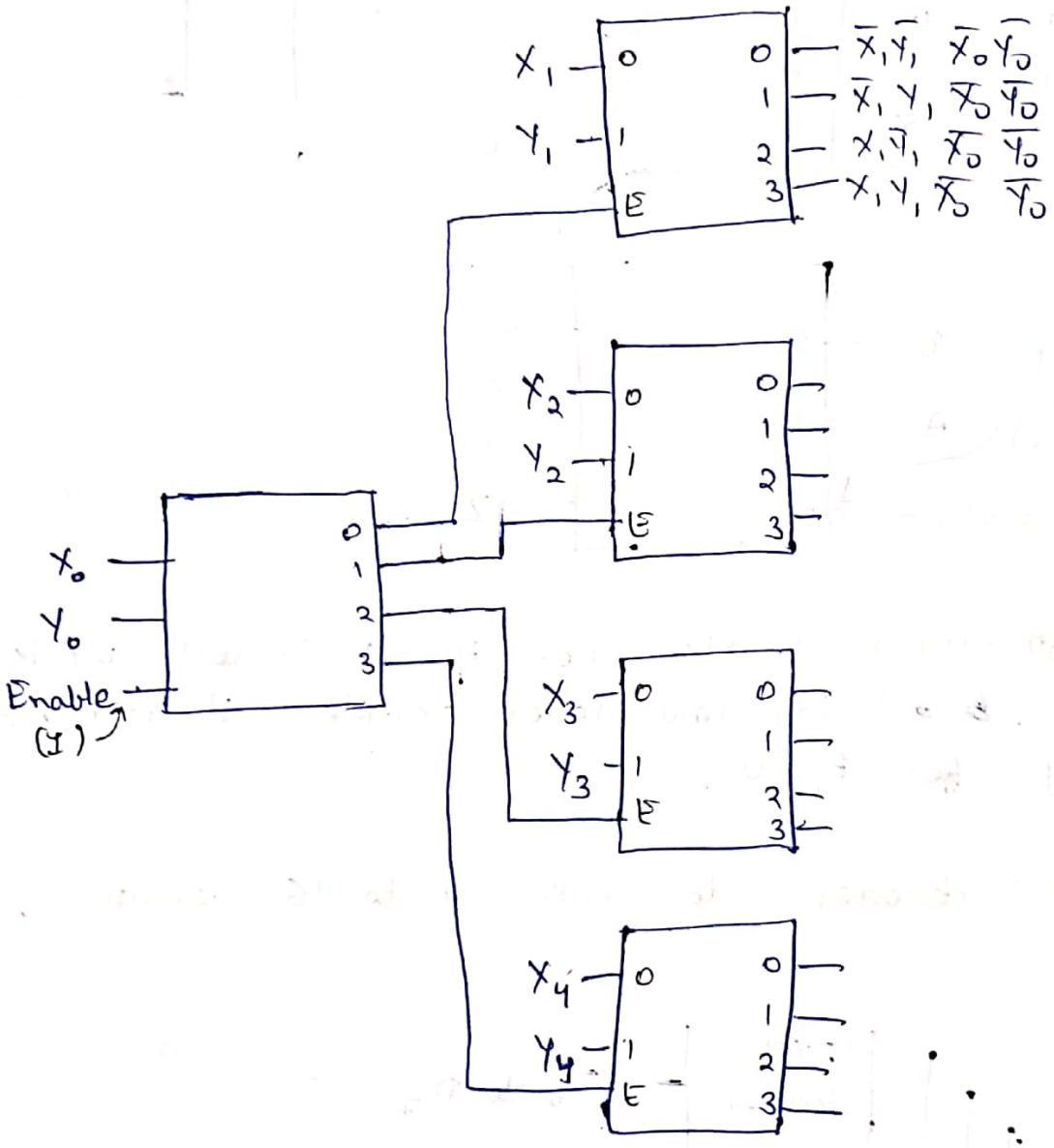


★ In High-level enable, For $E_0 = 1$, it will work normally, & in low-level enable it will work normally for $E_0 = 0$.

★ Use 3 to 8 decoder to make 4 to 16 decoders.



Assignment: Design a 4-to-16 decoder using 2-to-4 decoder only.



Encoders

- Inverse of Decoder.
- At any one time, only one input line has a value of 1.
- 2^n - to - n encoders. (i.e., max inputs $= 2^n$).

Eg:- (a) 8 - to - 3 Binary Encoder.

Inputs = I_0 to I_7

Outputs = ~~x, y, z~~ y_2, y_1, y_0

Then, ~~x~~ $y_2 = I_4 + I_5 + I_6 + I_7$

$y_1 = I_2 + I_3 + I_6 + I_7$

$y_2 = I_1 + I_3 + I_5 + I_7$

(b) ~~10~~ 10 : 4 encoder : Decimal to BCD.

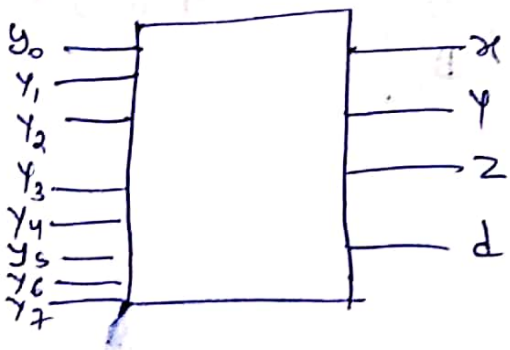
Priority Encoder.

- A priority scheme is assigned to the input ~~values~~ lines so whenever more than one input line is asserted at any time, the output is determined by the input line having the highest priority.

i.e.,

- higher - order input has more priority over lower order.

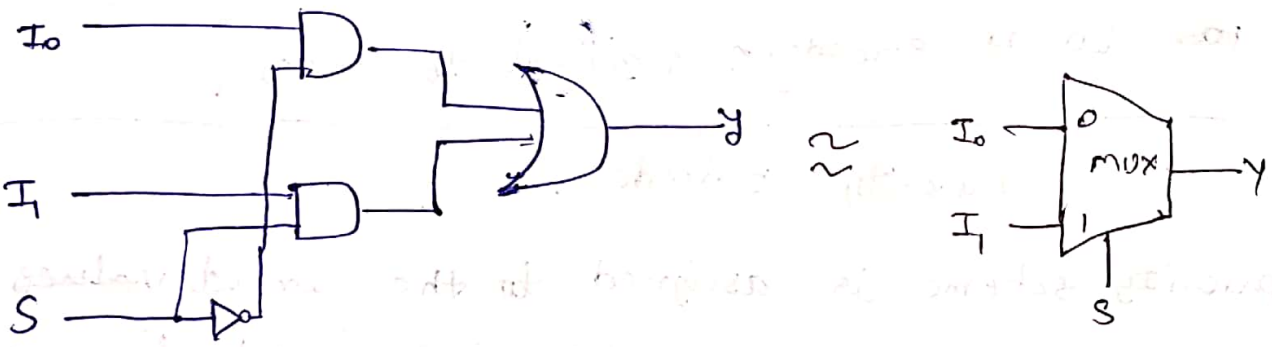
- It also has 'valid - bit' d. d = 0 (output not active)
  d = 1 (active output)



Multiplexer.

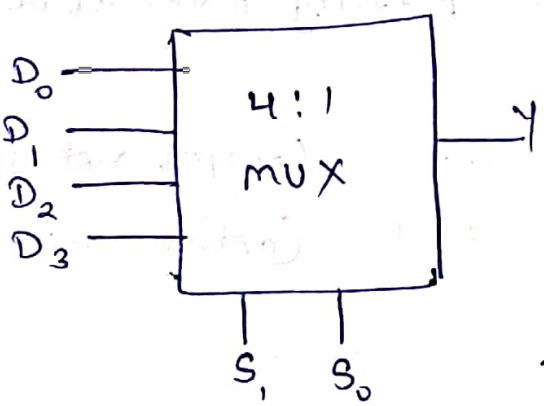
- 2^n inputs → 1 output.
  n = no. of selections. (So, n = selection lines)
- Also called data-selector.
- Allows conditional transfer of data.
- Sometimes called a mux.
  → Select one out of several bits
  → Some inputs used for selection.
  → Also can be used to implement logic.

Eg :- 2:1 MUX.



$Y = I_0\bar{S} + I_1 S$

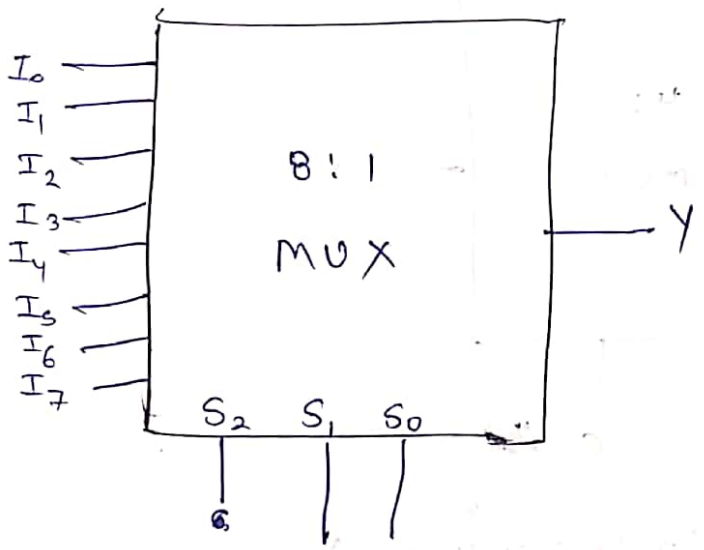
Eg :- 4:1 MUX.



S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$Y = S_1'S_0'D_0 + S_1'S_0D_1 + S_1S_0'D_2 + S_1S_0D_3$

Eg:- 8:1 MUX.



→ Can be used to implement combinational logic circuits.
→ Can replace several SSI gates.

Implementing Boolean Function using MUX :-

(1) n variable :- $2^{n-1} : 1$ MUX.
[∵ 1 variable is used as input & (n-1) as selection-lines]

(2) Input-variable is complemented in min-terms 0 to $2^{n-1}-1$ & uncomplemented in rest.

Eg:- $F(A, B, C) = \Sigma(1, 3, 5, 6)$

$2^{n-1} : 1 = 2^{3-1} : 1 = 4 : 1$

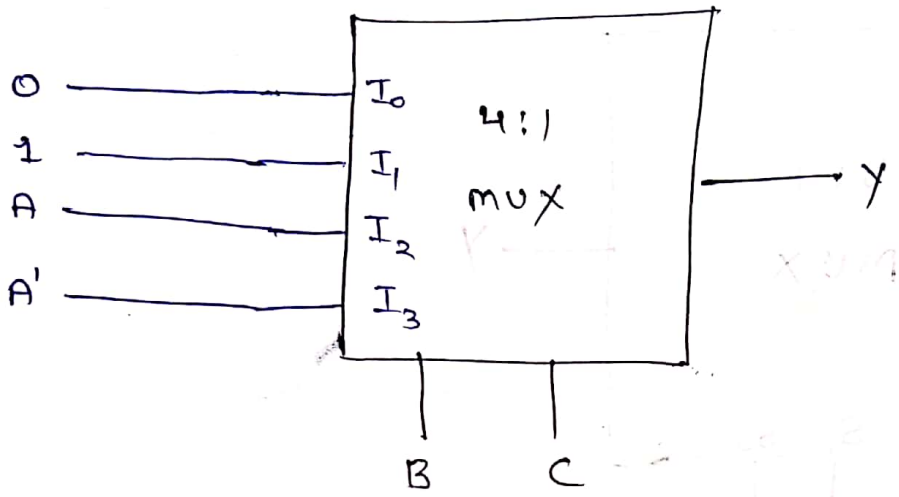
A → input
B, C → Selection-lines.

	I_0	I_1	I_2	I_3
A'	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	A'

≈

	I_0	I_1	I_2	I_3
A'	$B'C'$	$B'C$	BC'	BC
A	$B'C'$	$B'C$	BC'	BC

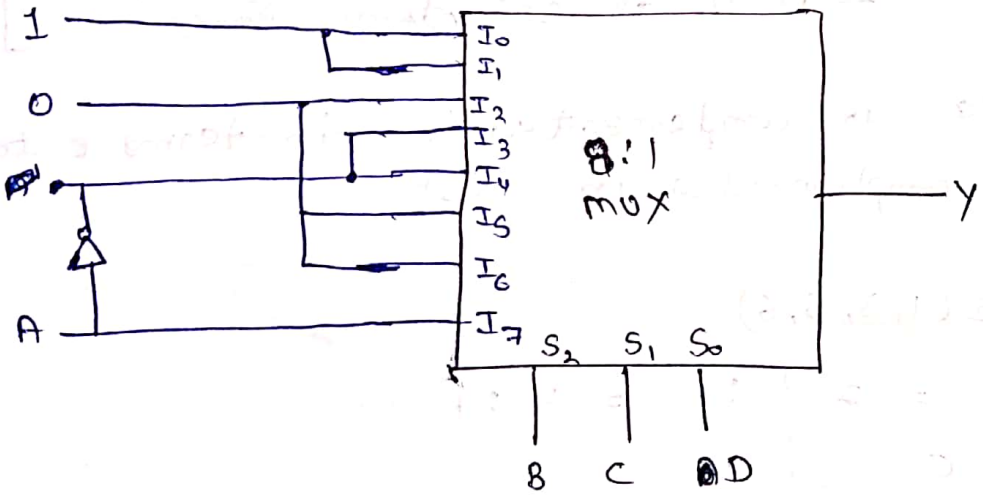
So, $Y = A'B'C + A'BC + AB'C + ABC'$



Eg :- $F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$

Input — Selection :-

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A'	(0)	(1)	2	(3)	(4)	5	6	7
A	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	A'	A'	0	0	A



Eg:- Implement $F(A,B,C,D) = \Sigma(0,1,3,4,8,9,15)$ using 4:1 MUX.

Solution:-

A, B → Inputs

C, D → Selection lines.

	I_0	I_1	I_2	I_3
$A'B'$	(0)	(1)	2	(3)
$A'B$	(4)	5	6	7
AB'	(8)	(9)	10	11
AB	12	13	14	(15)

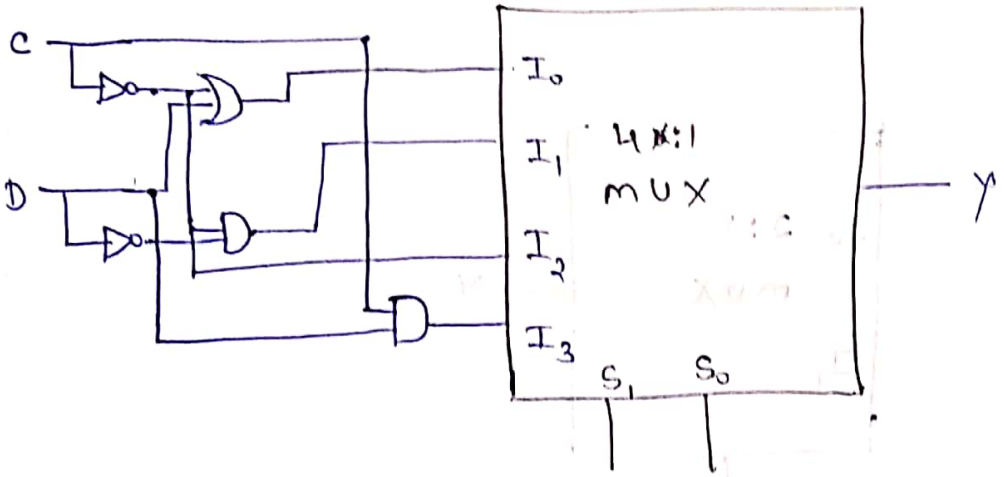
(or)

	$A'B'$	$A'B$	AB'	AB
	I_0	I_1	I_2	I_3
$C'D'$	(0)	(1)	(8)	12
$C'D$	(1)	5	(9)	13
CD'	2	6	10	14
CD	(3)	7	11	(15)
		$C'D'$	C'	CD

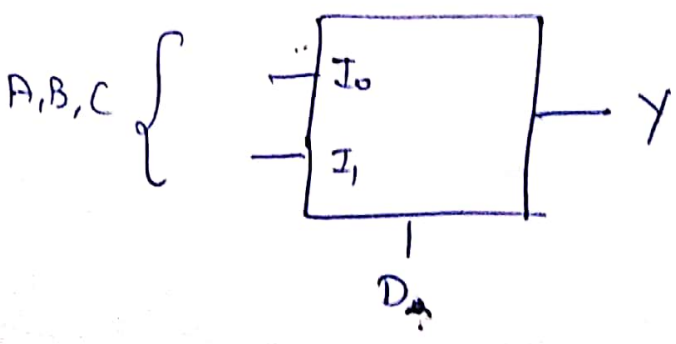
$I_0 = C' + CD$

$= C' + D$

S_0,



→ Using 2:1 MUX :-

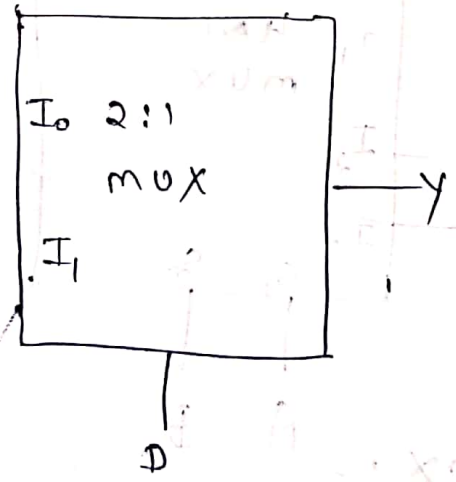


	D'	D
	I_0	I_1
$A'B'C'$	0	1
$A'B'C$	2	3
$A'BC'$	4	5
$A'BC$	6	7
$AB'C'$	8	9
$AB'C$	10	11
ABC'	12	13
ABC	14	15

$I_0 = A'C' + AB'C' = (A' + AB')C' = \boxed{(A' + B')C'} = \cancel{A'C' + B'C'}$

$I_1 = A'B' + AB'C' + ABC$

$= (A' + AC')B' + ABC = \boxed{(A' + C')B' + ABC}$

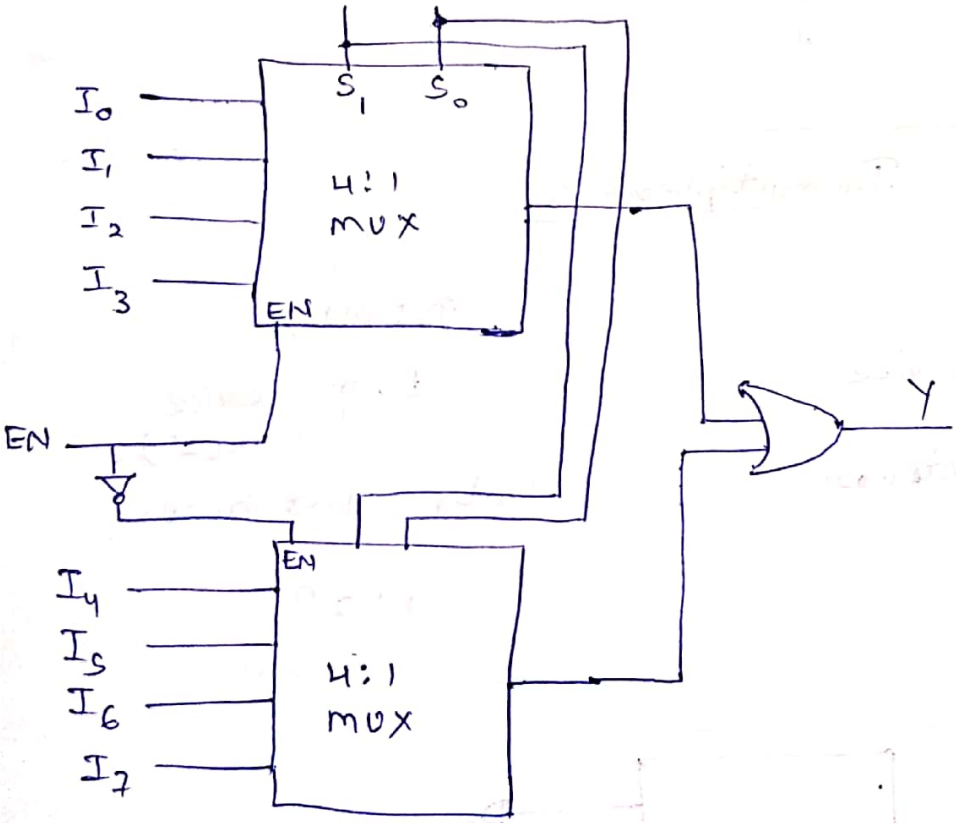


Multiplexer Tree

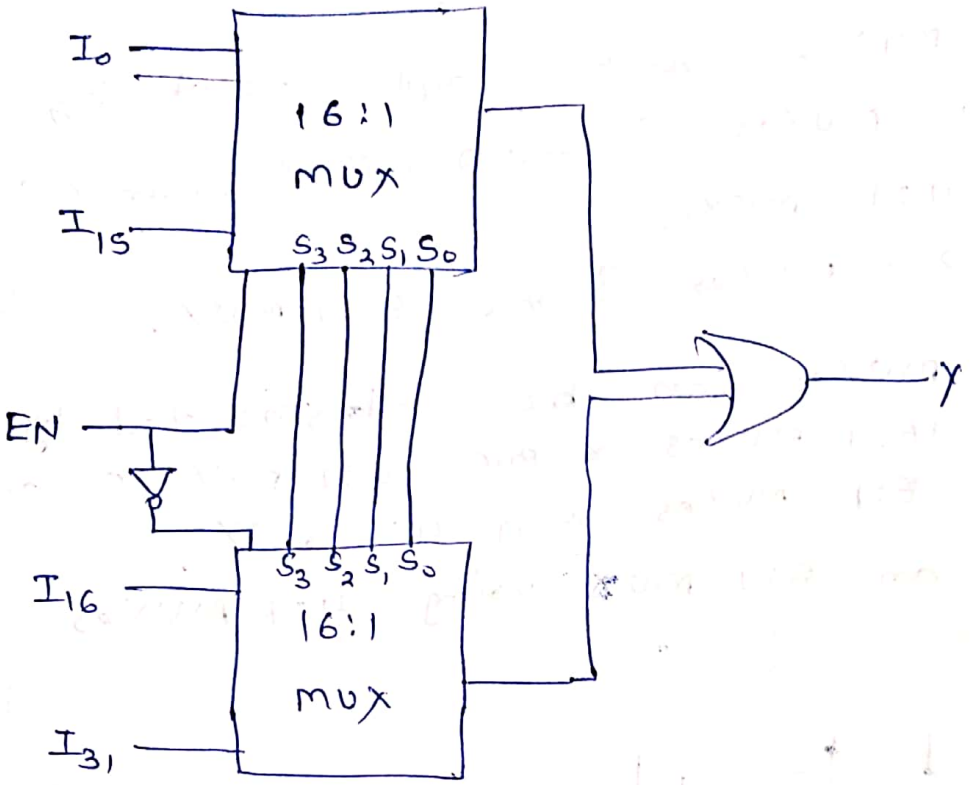
- A larger MUX can be implemented using a tree of smaller MUXes.
- A 16:1 MUXes can be implemented by
  - → Two 8:1 MUXes & one 2:1 MUX or an OR gate
  - → Five 4:1 MUXes
  - → Eight 2:1 MUXes & one 8:1 MUX.
- A 32:1 MUXes can be constructed by using
  - → Two 16:1 MUXes & one 2:1 MUX or an OR gate
  - → Four 8:1 MUXes & a 4:1 MUX

Eg:- Design an 8:1 MUX using 4:1 MUXes.

Solution:-

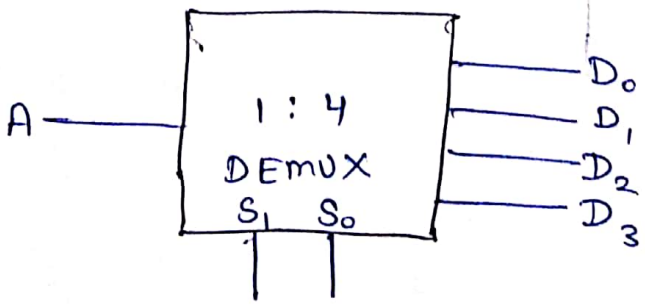


Eg: Design a 32:1 MUX using 16:1 MUXes.



Demultiplexers

	MUX	DEMUX
(1)	m : 1 device	1 : m device ↳ max(2^n)
(2)	Data Selectors	Data distributors
(3)	2^n : 1	1 : 2^n

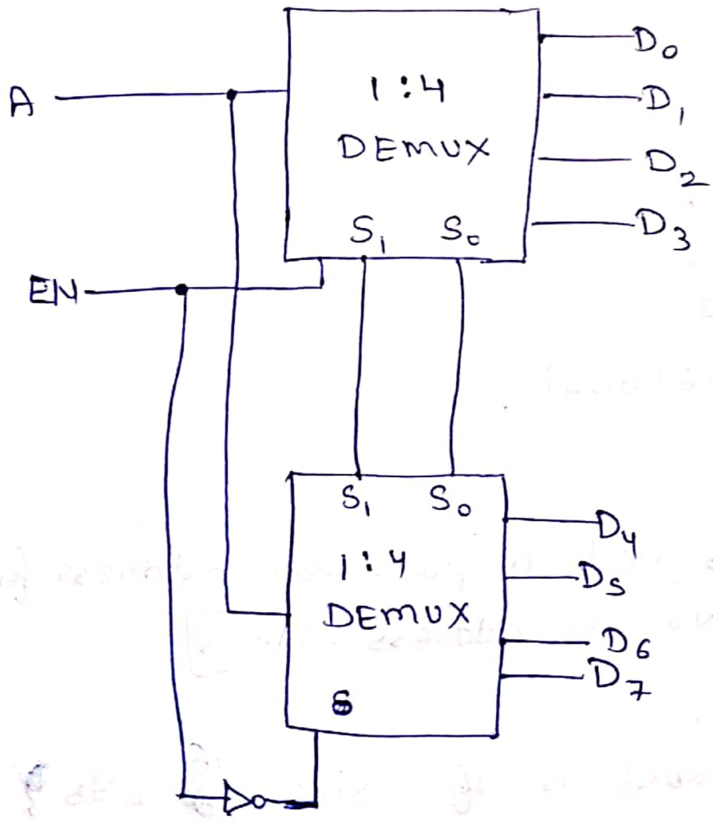


~~A Selection lines~~.

A	Select S_1	S_0	Output D_0	D_1	D_2	D_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Eg:- Design a 1:8 DEMUX using 1:4 DEMUXes.

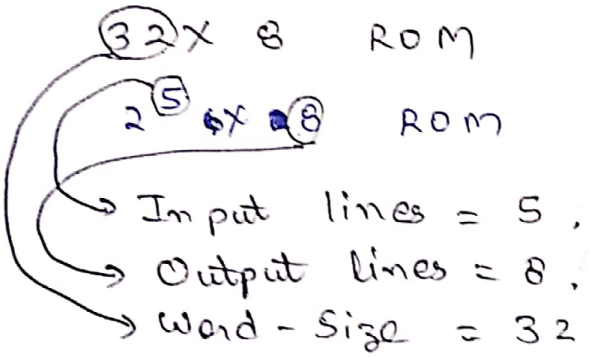
Solution:-



Scanned by CamScanner

Combinational Logic Design using ROM Array.

- ROM is a memory unit that performs only the read operation.
- Binary information stores in ROM is permanent & is created during fabrication process.
- ROM is a programmable logic device (PLD)
- A word is the basic unit that moves in & out of a memory.
- Word = group of bytes. (= 8 bits)



Eg :- 1024 (words) × 16 (bits).

$2^{10} \times 16$.

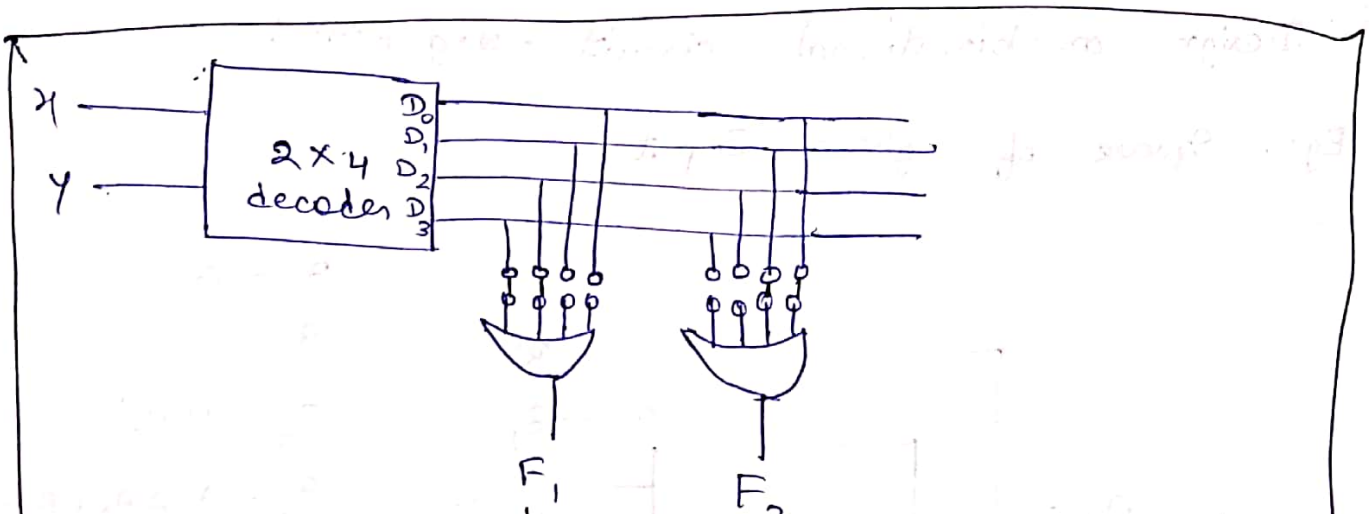
Each word is assigned a particular address from 0 to $2^K - 1$. [K = No. of address-lines]

Input lines = 10.

~~Output lines~~ Each word is of size 16 bits.

Eg :- $F_1(x,y) = \Sigma(2,3)$

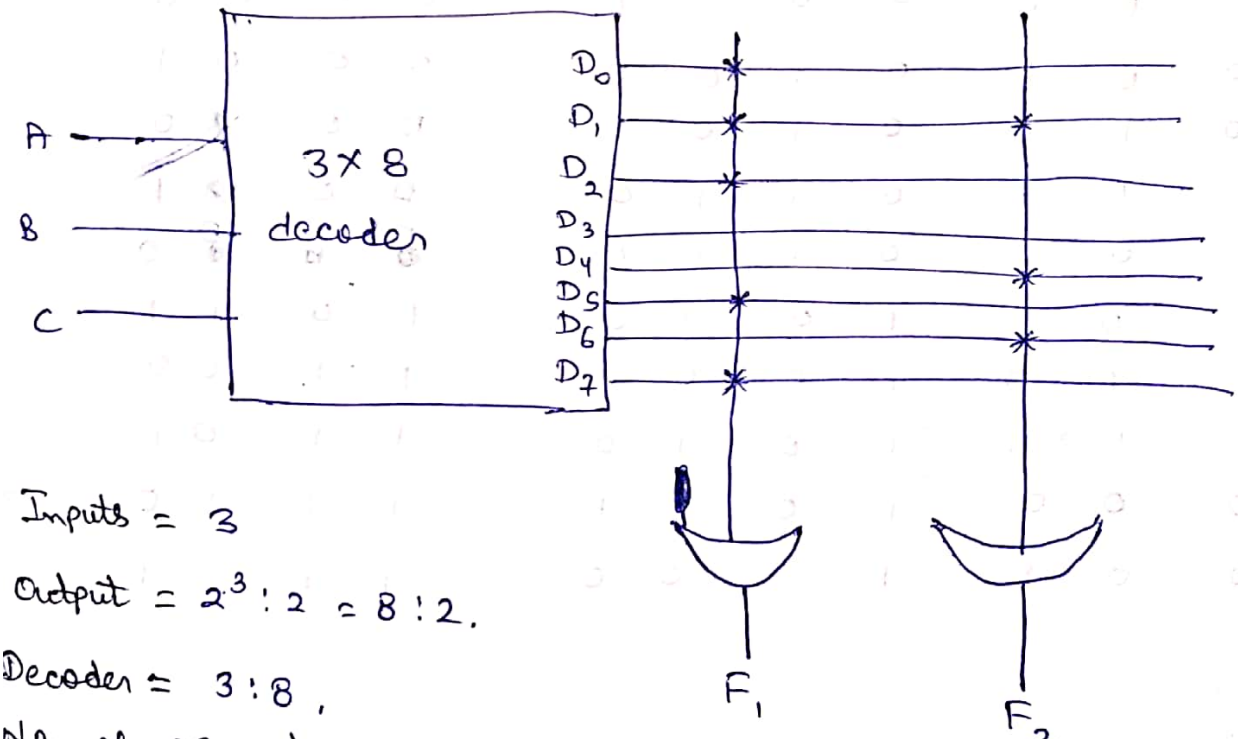
$F_2(x,y) = \Sigma(0,1)$.



Specification of ROM =	$2^2 : 2$ $4 : 2$ ROM

Eg:- $F_1(A, B, C) = \Sigma(0, 1, 2, 5, 7)$,

$F_2(A, B, C) = \Sigma(1, 4, 6)$



Inputs = 3

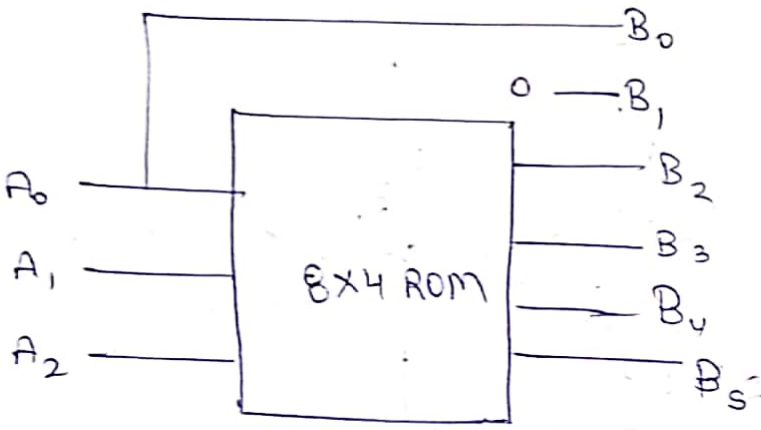
Output = $2^3 : 2 = 8 : 2$.

Decoder = 3 : 8 ,

No. of OR gates = 2 ,

Design combinational circuit using ROM:-

Eg:- Square of given Input



$B_0 = A_0$

$B_1 = 0$

$B_2 = A_1 A_0'$

$B_3 = A_2' A_1 A_0 + A_2 A_1' A_0$

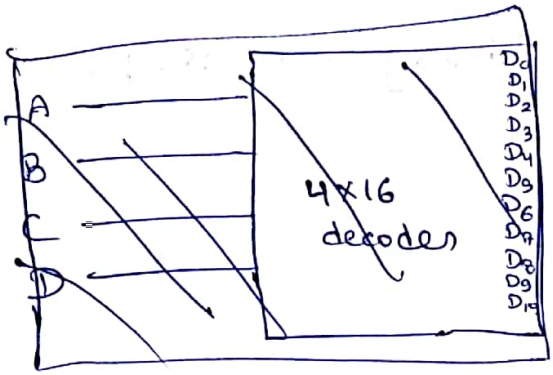
$B_4 = A_2 A_1' + A_2 A_1 A_0$

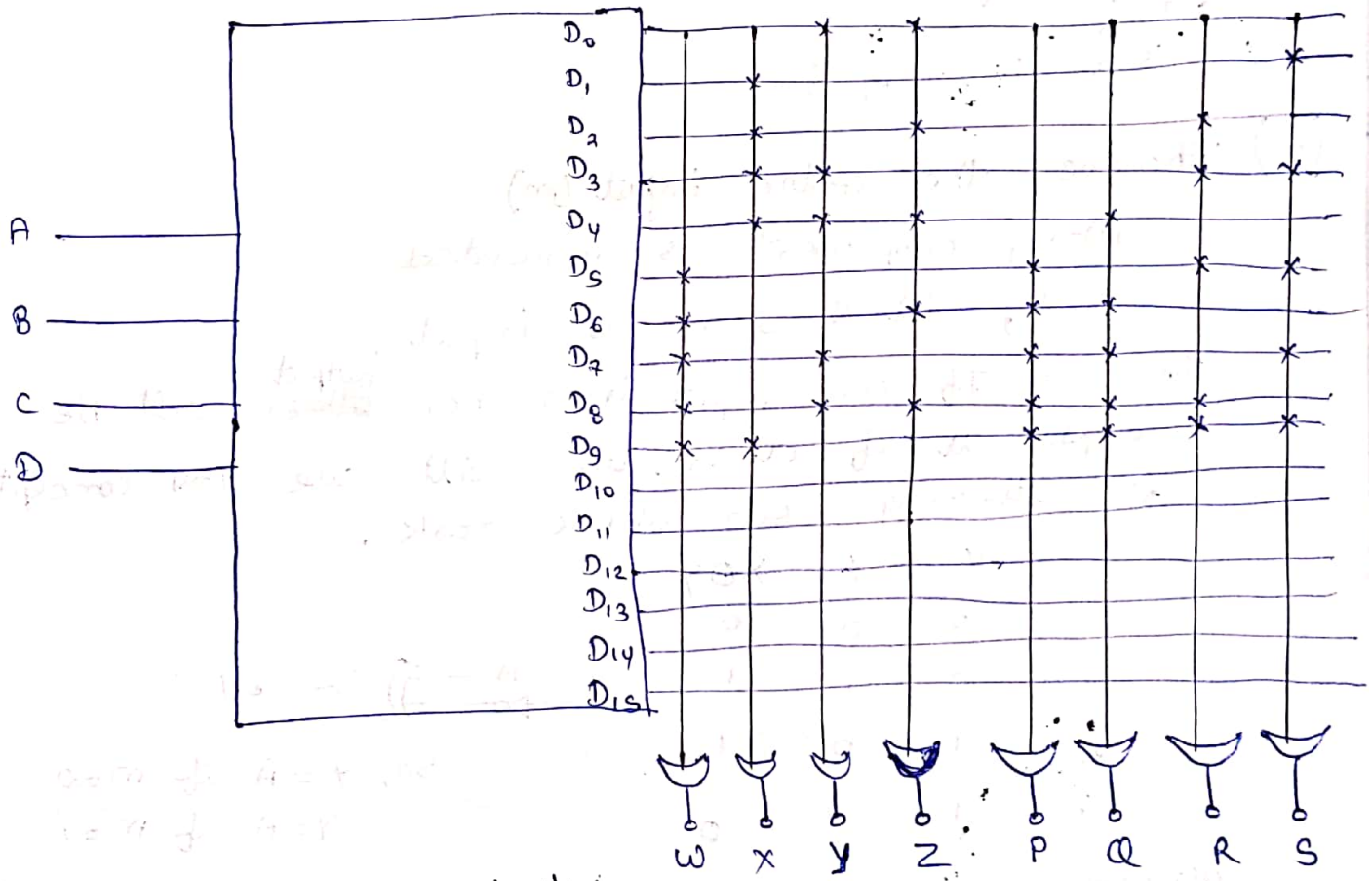
$B_5 = A_2 A_1$

B_0 & B_1 are redundant outputs.

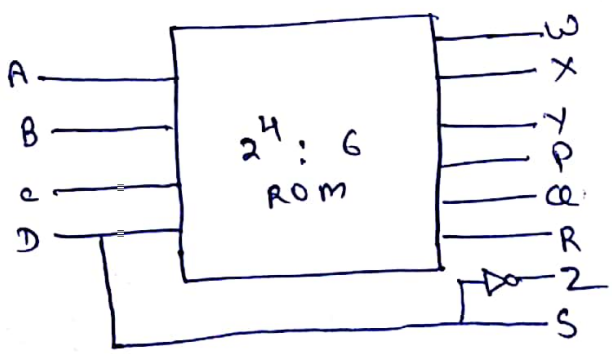
Eg:- Design a code converter that converts BCD to excess-3 & 2421 using ROM array.

BCD				Xs-3				2421			
A	B	C	D	W	X	Y	Z	P	Q	R	S
0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	1
0	0	1	0	0	1	0	1	0	0	1	0
0	0	1	1	0	1	1	0	0	0	1	1
0	1	0	0	0	1	1	1	0	1	0	0
0	1	0	1	1	0	0	0	1	0	1	1
0	1	1	0	1	0	0	1	1	1	0	0
0	1	1	1	1	0	1	0	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0
1	0	0	1	1	1	0	0	1	1	1	1





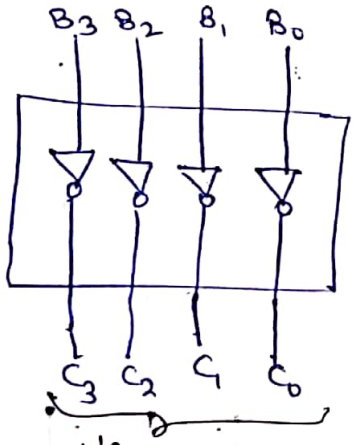
→ Removing Redundant Outputs :-



Assignment : 4-bit multiplier & 2-bit multiplier,

1's complement of a 4-bit binary number :-

(i) $B = B_3 B_2 B_1 B_0$



1's compliment of B.

(ii) Whenever the control input (m)

$m=0$, complement is generated,

$m=1$, it is same as input,

In XOR, If one-input is 1, then ~~other~~ output, will be complement of other. We will use this concept for attaining our above task.

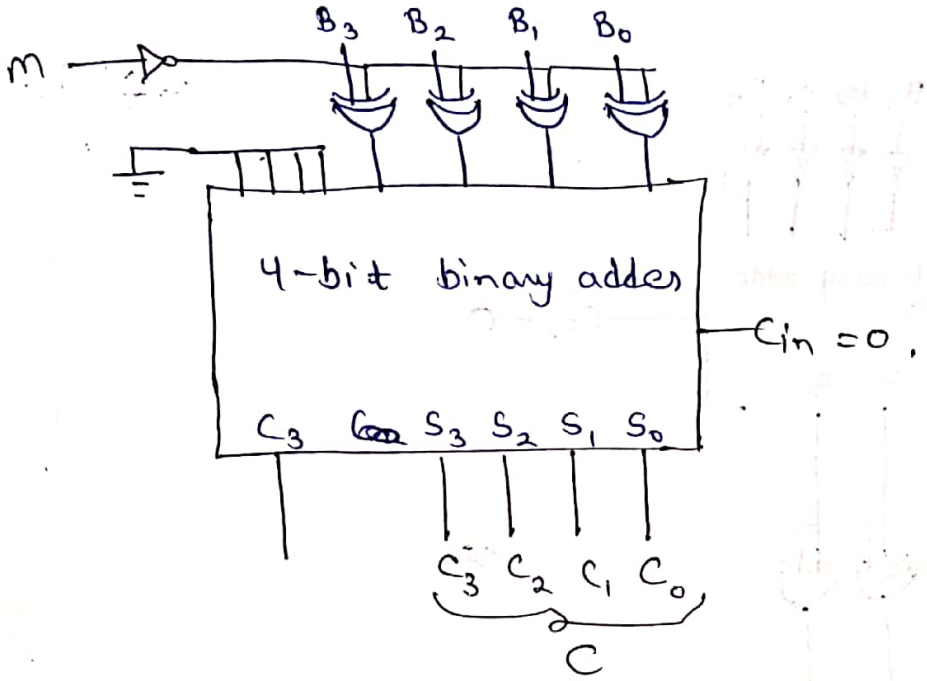
X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

So, $Y = \bar{A}$ if $m=0$

$Y = A$ if $m=1$

Using :-

4-bit Binary adder :-



(iii) Let's Design circuit for 9's complement :-

9's comp. of 4 = 5

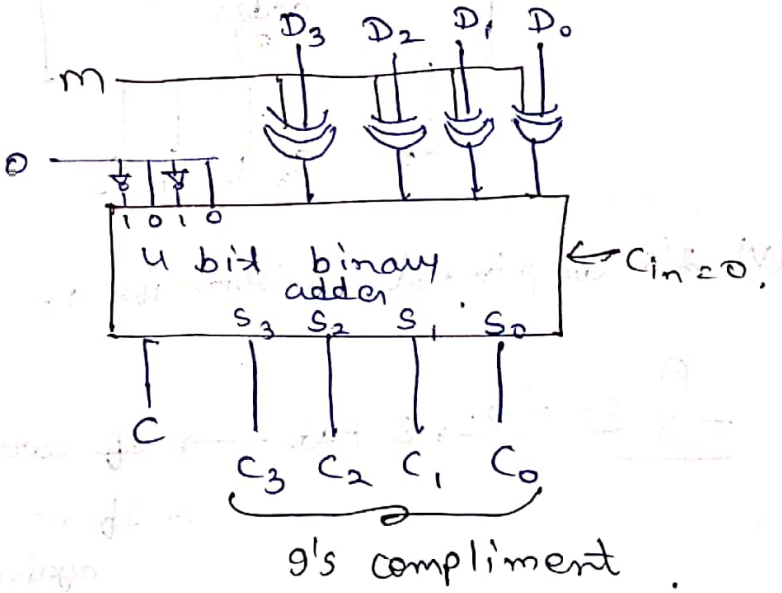
0100

1011 ⟶ 1's comp.

1010 ⟶ Add 10.

1 | 0101 → Discard Carry

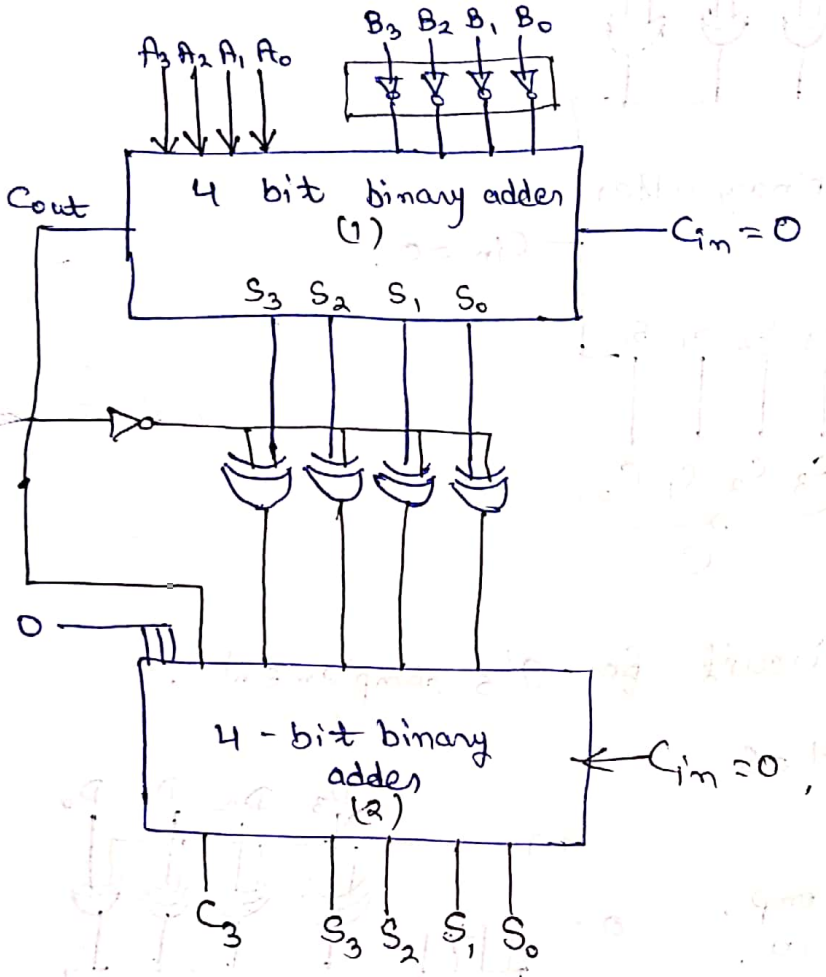
0101 = 5



(iv) 4-bit Subtractor using Adders :-

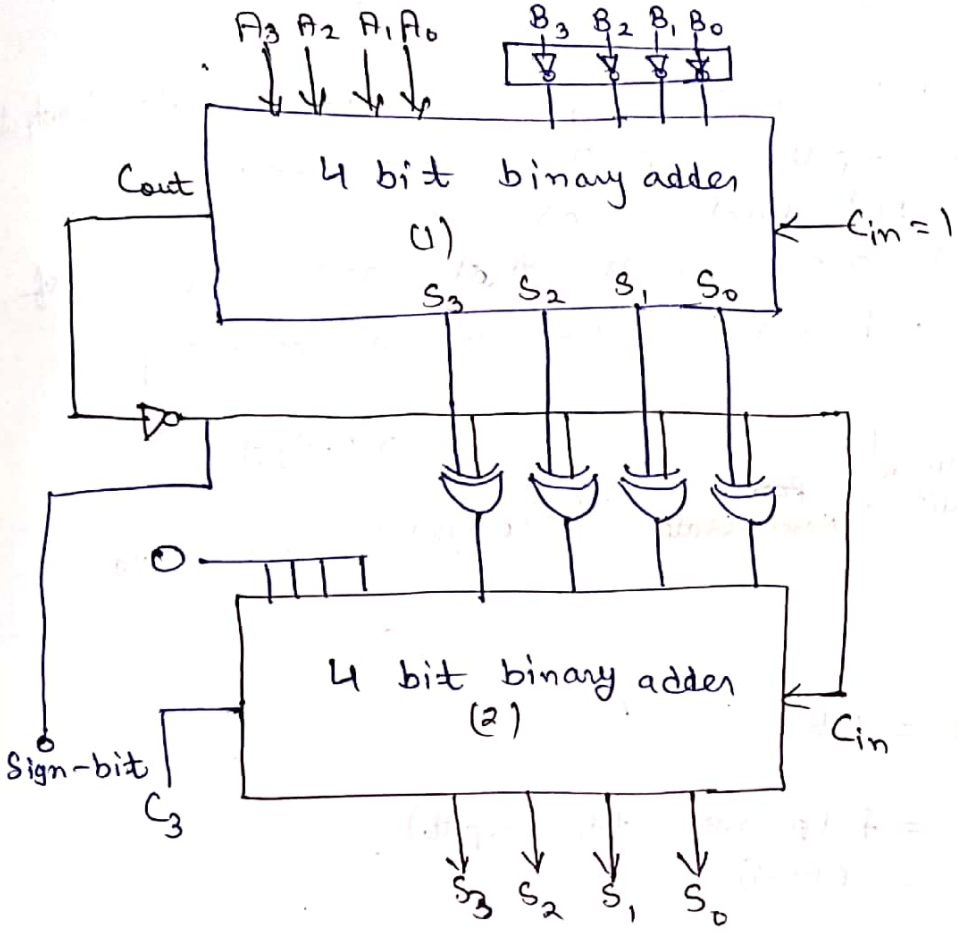
→ Using 1's compliment.

P. T. O.



(V) 2's complement Subtraction :-

$$\begin{array}{l} A \\ -B \end{array} \xrightarrow{\text{2's comp.}} \text{\& Add} \longrightarrow \text{If carry generated, discard, \& If no carry, ans is -ve \& take 2's compliment again.}$$



(vi) Design a SOP circuit that will generate an odd-parity for a 4-bit input.

Input				Output
A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

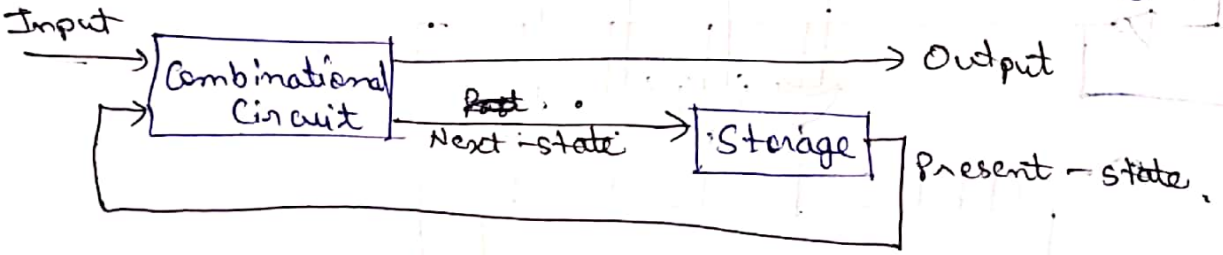
$F = \bar{A}D(B \oplus C) + \bar{A}\bar{D}(\overline{B \oplus C}) + A\bar{B}(C \oplus D) + AB(\overline{C \oplus D})$

$= \bar{A}\bar{D}(A \oplus B \oplus C) + A(B \oplus C \oplus D)$

$= \boxed{(A \oplus B \oplus C \oplus D)}$

→ Introduction to Sequential Design ←

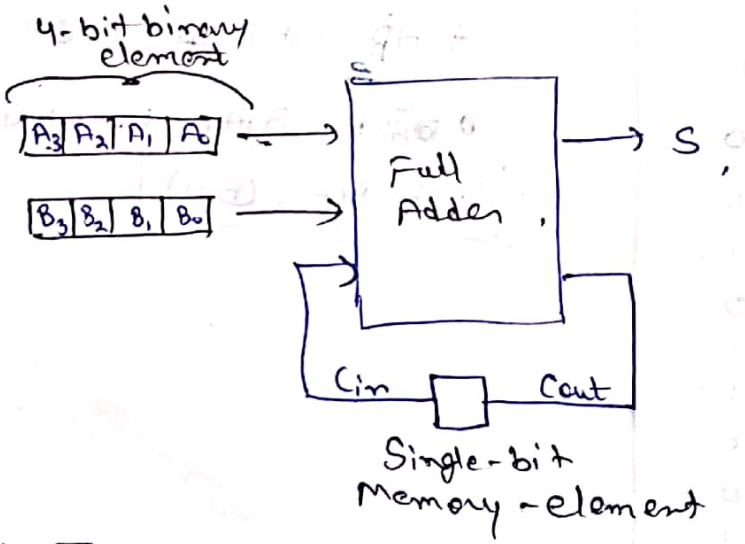
- Output depends upon both past & present inputs (& outputs)
- Remembers past inputs & past circuit-state
- Outputs are "fed back" as new inputs.
- Storage elements are circuits that are capable of storing binary information : known as memory.



- Present-state = $Q(t)$
- Next-state = f (present-state, input)
  = $Q(t+1)$

Eg:- • Traffic-light
• ATM
• Vending Machine.

Serial or Sequential Adder :-
(Already discussed combinational adder).

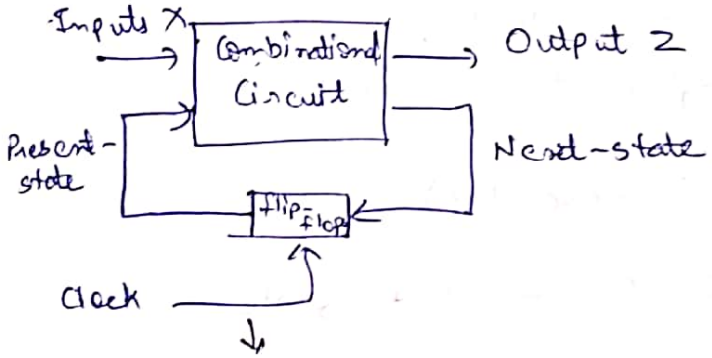


→ Time is more, but hardware is less.
- 1 Full-adder,
- 2 4-bit memory,
- 4 clocks to get the output.
- The 1-bit memory element

Sequential Circuits

Synchronous

Behaviour of the circuit depends on the input signal at discrete instances of time (also called clocked)



- Each flip-flop can store one bit of information : 0 or 1.

Asynchronous

The behaviour of circuit depends on input signals at any instance of time & order of the inputs change.

- A combinatorial circuit with feedback.

Storage Elements (memory)

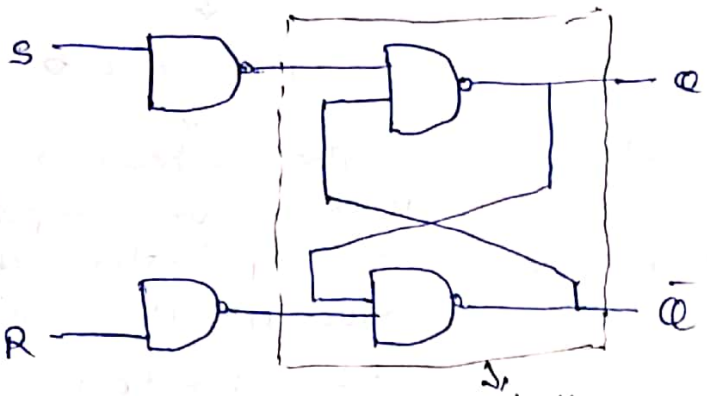
- Two main types: Latches – Asynchronous (without clock signal)
  Flip-Flops – Synchronous, (with clocks)

Latches :
- A binary storage element (0 or 1 – 1 bit info.)
- Built with (NAND, NOR, NOT).

Flip-Flop :
- Synchronous version of latch

→ In latch, output changes as soon as input changes, while in flip-flops, it is not so.

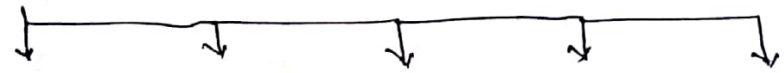
Basic Flip-Flop :-



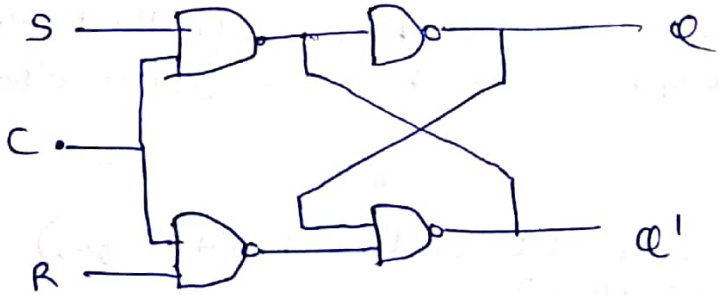
Basically, it is flip-flop circuit.

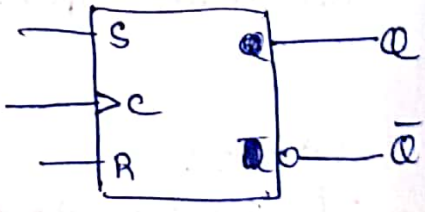
- Two Outputs : Q = Normal Output,
  Q' – Complemented Output
- Two Inputs : Set (S) & Reset (R)
- Two-states : Q=1, Q'=0 ⇒ Set-state
  Q=0, Q'=1 ⇒ Clear-state

Types :



SR :-





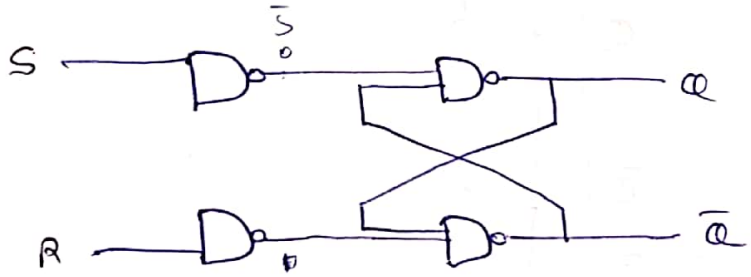
Characteristic-Table :

→ Logical properties of a flip-flop.
→ Q(t)
→ Q(t+1)

SR : Characteristic-Table

S	R	Q(t+1)	Operation
0	0	Q(t)	No change / Hold
0	1	0	Reset
1	0	1	Set
1	1	?	Undefined/Invalid

Let's verify above table:-



(i) When S=0, R=0,

Let Q(t) = 0 ⇒ $\bar{Q}$(t) = 1 ⇒ Q(t+1) = 0,

Then,

Let Q(t) = 1 ⇒ $\bar{Q}$ = 0 ⇒ Q(t+1) = 1.

So, There is no change in state.

(ii) When S=0, R=1,

Let Q(t) = 0 ⇒ $\bar{Q}$ = 1 ⇒ Q(t+1) = 0

Let Q(t) = 1 ⇒ $\bar{Q}$ = 1 ⇒ Q(t+1) = 0

(iii) When S=1, R=0, Q(t+1) = 1.

(iv) When S=1, R=1,

∵ One input to both Nand-gates is 0,

∴ Output of both must be 1,

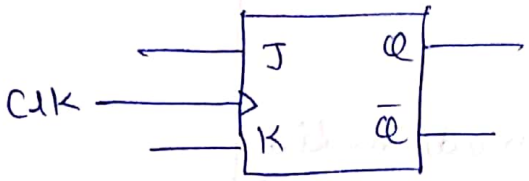
So, Q = 1 & $\bar{Q}$ = 1,

↓

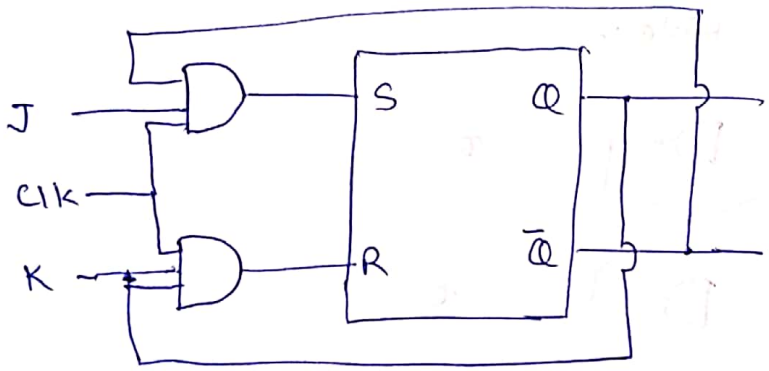
Invalid.

JK Flip-Flop

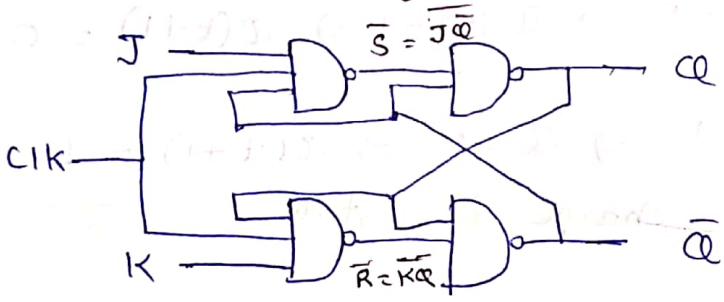
- It will avoid $S=R=1$.



$S = J\bar{Q}$
$R = KQ$



⇓ NAND-level Diagram.



Characteristic Table :- (Flip-Flop)

Inputs		Outputs
J_n	K_n	(Q_{n+1})
0	0	Q_n
0	1	0 (Reset)
1	0	1 (Set)
1	1	$\bar{Q}_n$ (toggle)

$S_n = J_n \bar{Q}_n$ & $R_n = K_n Q_n$

J_n	K_n	Q_n	$\bar{Q}_n$	S_n	R_n	Q_{n+1}	
0	0	0	1	0	0	$Q_n = 0$	$\rightarrow Q_n$
0	0	1	0	0	0	$Q_n = 1$	
0	1	0	1	0	0	$Q_n = 0$	$\rightarrow 0$
0	1	1	0	0	1	0	
1	0	0	1	1	0	1	
1	0	1	0	0	0	$Q_n = 1$	$\rightarrow 1$
1	1	0	1	1	0	1	
1	1	1	0	0	1	0	$\rightarrow \bar{Q}_n$

Drawback of JK - Flip-flop :-

→ RACE Around condition :

$t \rightarrow$ propagation delay of gates (NAND)

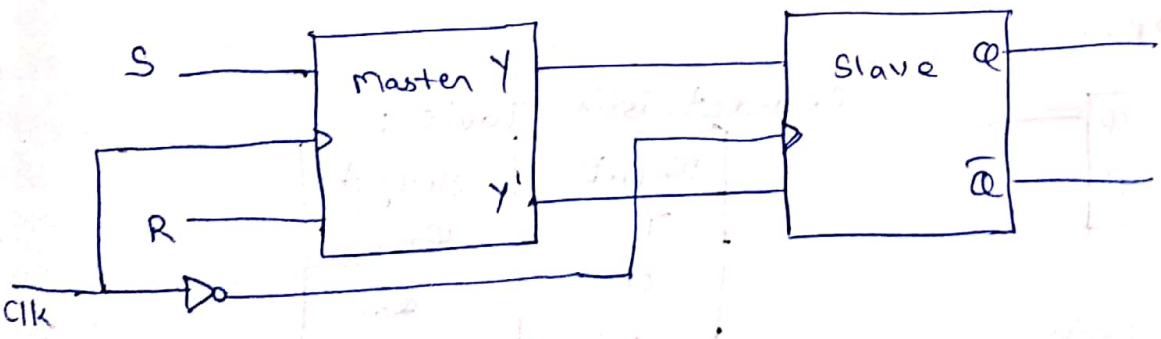
$T \rightarrow$ clock-pulse,

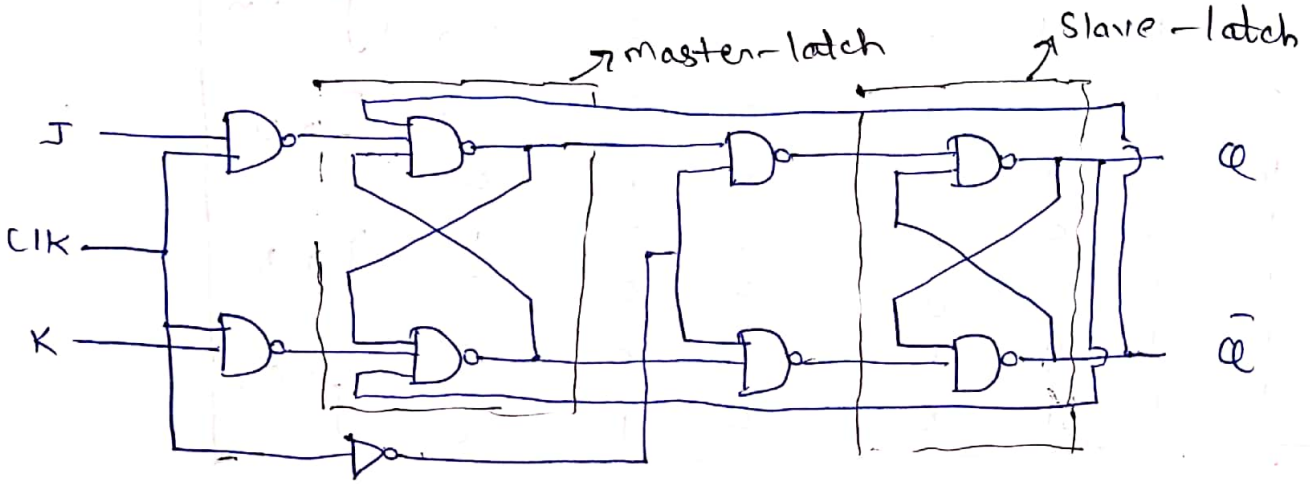
If $t << t$, after each time interval t, for $J=K=1$, our outputs repeatedly gets complemented in the same clock-pulse.

This condition is known as "Race condition."

So, at the end of time-pulse, output is not certain, To avoid this, master-slave was designed.

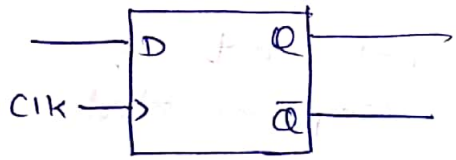
MASTER-SLAVE Flip-Flop :-





D - flip-flop :- (D stands for Data, or Delay)

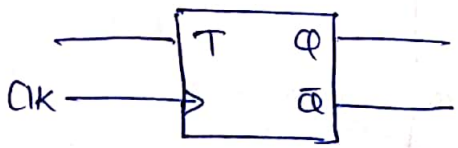
→ to transfer data
→ to delay data by one clock-pulse.



Characteristic Table :-

Input	Output
$D \equiv Q_n$	Q_{n+1}
0	0
1	1

T - flip-flop :-



Characteristic Table :-

Input	Output
T	Q_{n+1}
0	Q_n
1	$\bar{Q}_n$

Char. Eqn : $Q_{n+1} = T'Q_n + T\bar{Q}_n$

Excitation - table for flip-flops :-

- to find input - combinations for transition from particular present - state to particular next - state.

→ SR :-

Characteristic Table		
Input S	R	Output Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	?

⟹

Excitation - Table			
Input Q_n	Q_{n+1}	Output S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

→ JK

Characteristic Table		
Input J	K	Output Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

⟹

Excitation Table			
Input Q_n	Q_{n+1}	Output J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

→ D

Characteristic Table	
Input D_n	Output Q_{n+1}
0	0
1	1

⟹

Excitation Table		
Input Q_n	Q_{n+1}	Output D_n
0	0	0
0	1	1
1	0	0
1	1	1

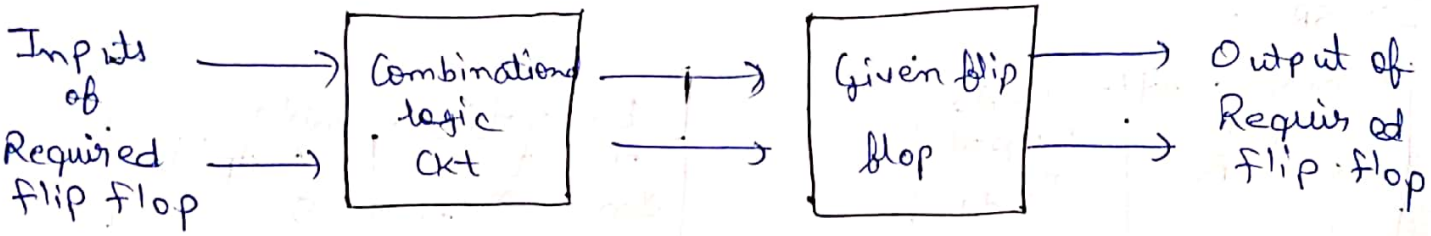
→ T

Characteristic Table	
Input T	Output Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

Excitation Table		
Input Q_n	Q_{n+1}	Output T
0	0	0
0	1	1
1	0	1
1	1	0

Flip-Flop Conversion :-

→ Given Flip-Flop
→ Required Flip-Flop.



Eg :- 1) SR to JK flip-flop

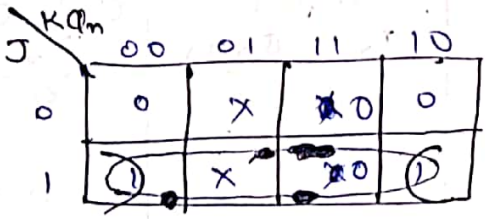
JK Inputs		Present State	Next State		
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X 0	1
1	0	0	1	1	X 0
1	0	1	1	X	0
1	1	0	1	1	X 0
1	1	1	0	X 0	1

From char. table of JK

From Exc. table of SR

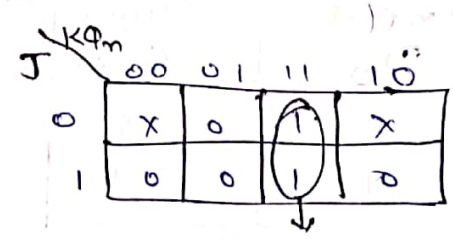
By K-Map :-

For S :-



$S = J\overline{Q}_n$

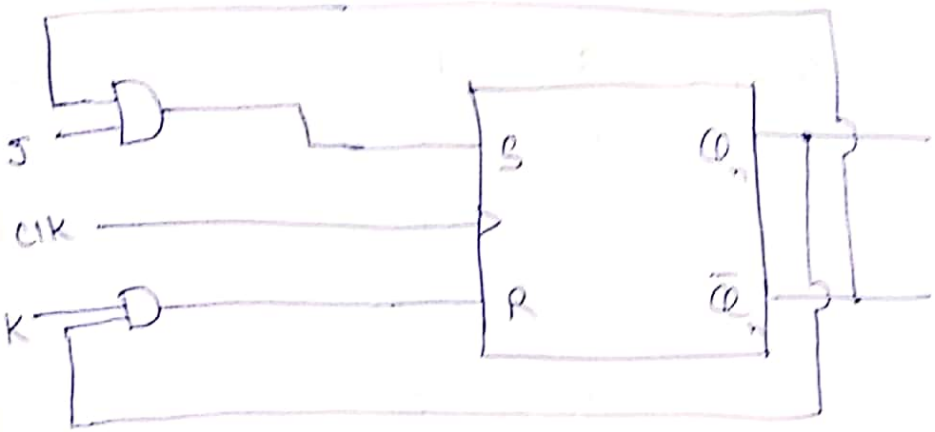
For R



$R = KQ_n$

So,

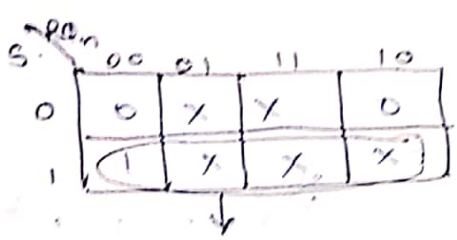
Logic - Diagram :-



Eg:- (2) JK to SR

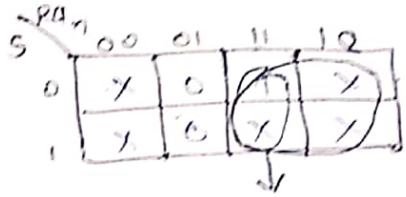
S	R	Q_n	Q_{n+1}	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	0	?	X	X
1	1	1	?	X	X

For J



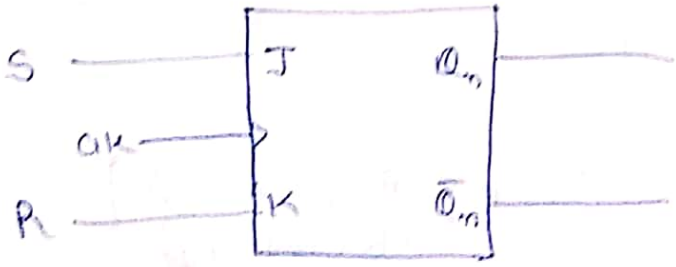
$J = S$

For K



$K = R$

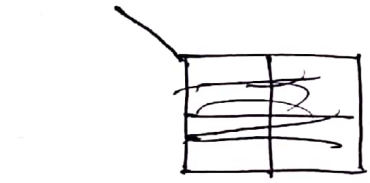
Logic - Diagram :-

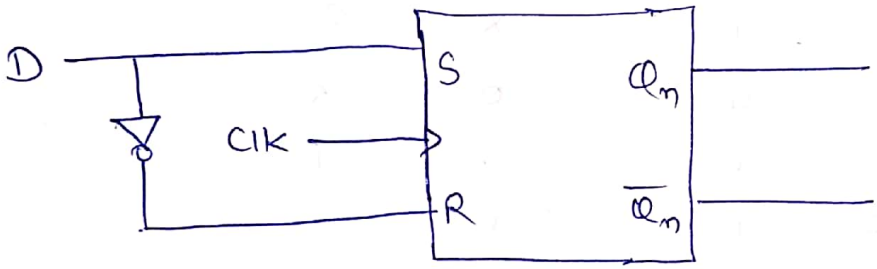


(3) SR to D flip flop :-

D	Q_n	Q_{n+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

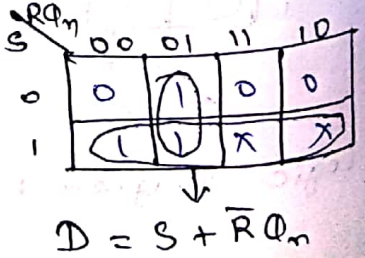
So, $S = D$ & $R = \bar{D}$





(4) D to SR flip-flop :-

S	R	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	?	X
1	1	1	?	X

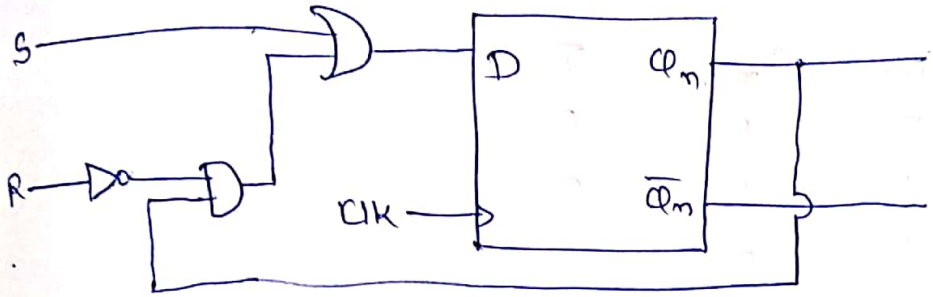


$D = S + \bar{R}Q_n$

So, ~~$D = \bar{S}\bar{R}Q + S\bar{R} = \bar{R}(\bar{S}Q + S)$~~ $= \bar{R}(Q+S)$ ~~Q~~

$D = S + \bar{R}Q_n$

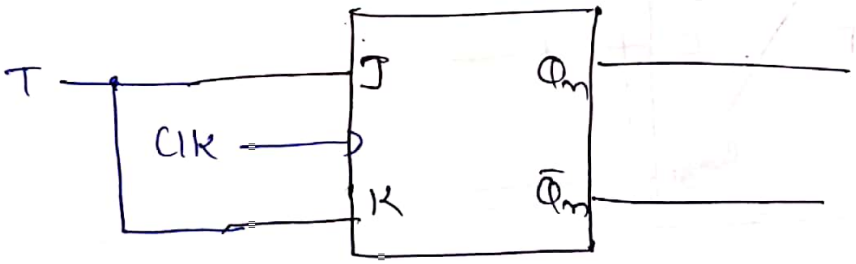
Logic Circuit :-



(5) JK ~~to~~ T flip-flop :-

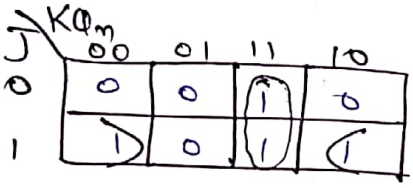
T	Q_n	Q_{n+1}	J	K
0	0	0	0	x
0	1	1	x	0
1	0	1	1	x
1	1	0	x	1

$J = \bar{Q}_n T$ & $K = Q_n T$



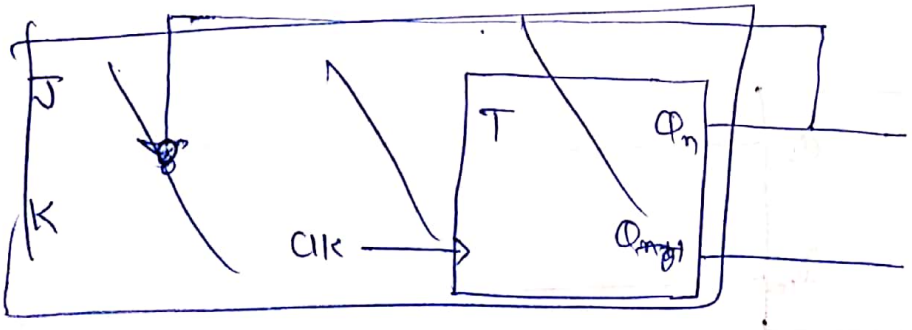
(6) T to JK :-

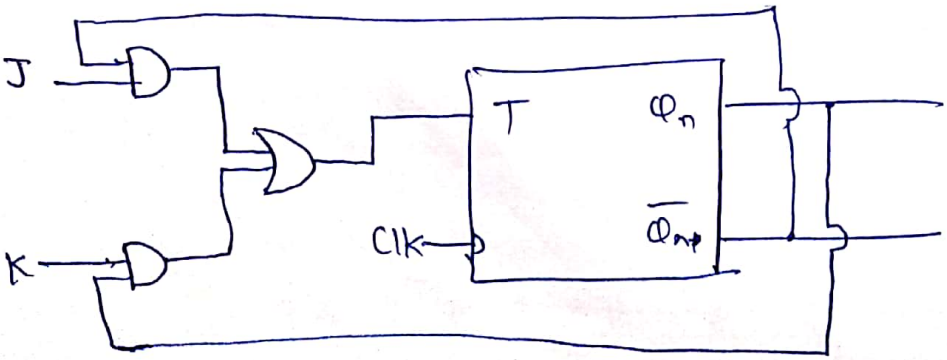
J	K	Q_n	Q_{n+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1



So, $T = KQ_n + J\bar{Q}_n$

Logic – Diagram :-

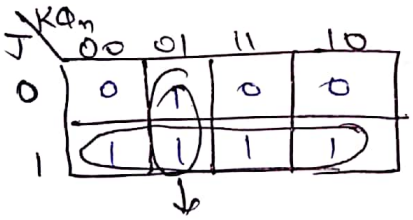




(7) D to JK :-

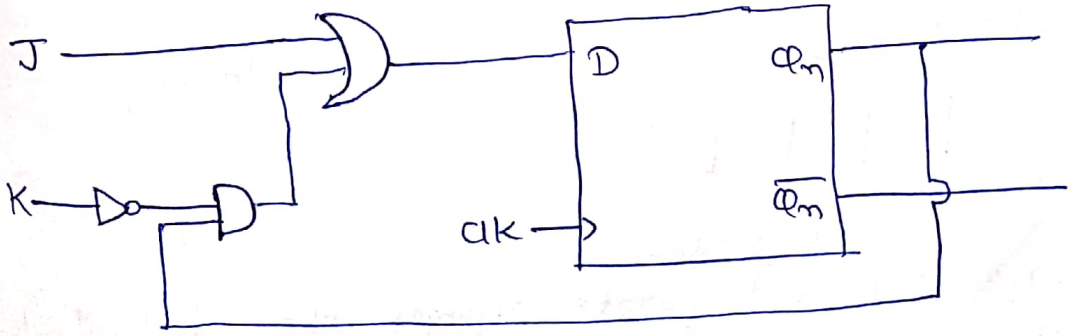
J	K	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	1

For D :-



$D = \bar{K} Q_n + J$

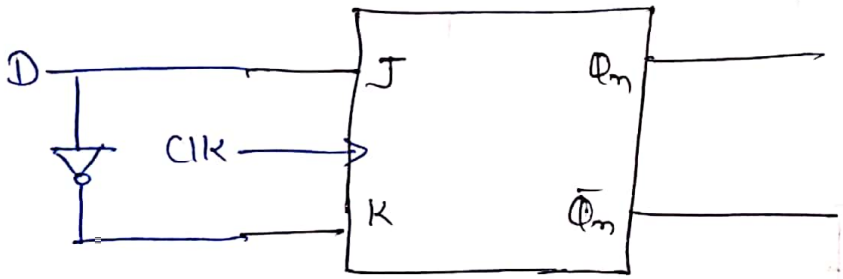
Logic-Diagram :-



(8) JK to D :-

D	Q_n	Q_{n+1}	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

So, $J = D$ & $K = \overline{D}$



→ Level - triggered :-

↳ triggering is done by changing clock-pulse.

Edge triggered flip-flop :-

↳ Clock - changes - state.

↳ state changes from 0 to 1 or 1 to 0.

Clock-pulse :

Types :-

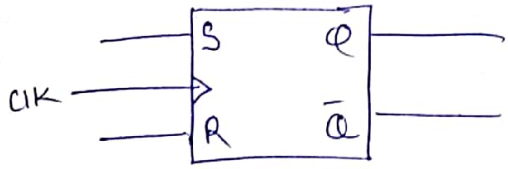
↳ +ve edge triggered (leading edge) (state changes when clock-pulse goes from 0 to 1)

↳ -ve edge triggered (when pulse goes from 1 to 0).

↳ (falling edge)

Edge - triggered SR flip-flop :-

(1) +ve edge triggered SR f/f :-

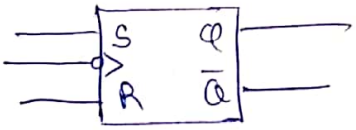


★ S and R are called synchronous control inputs.

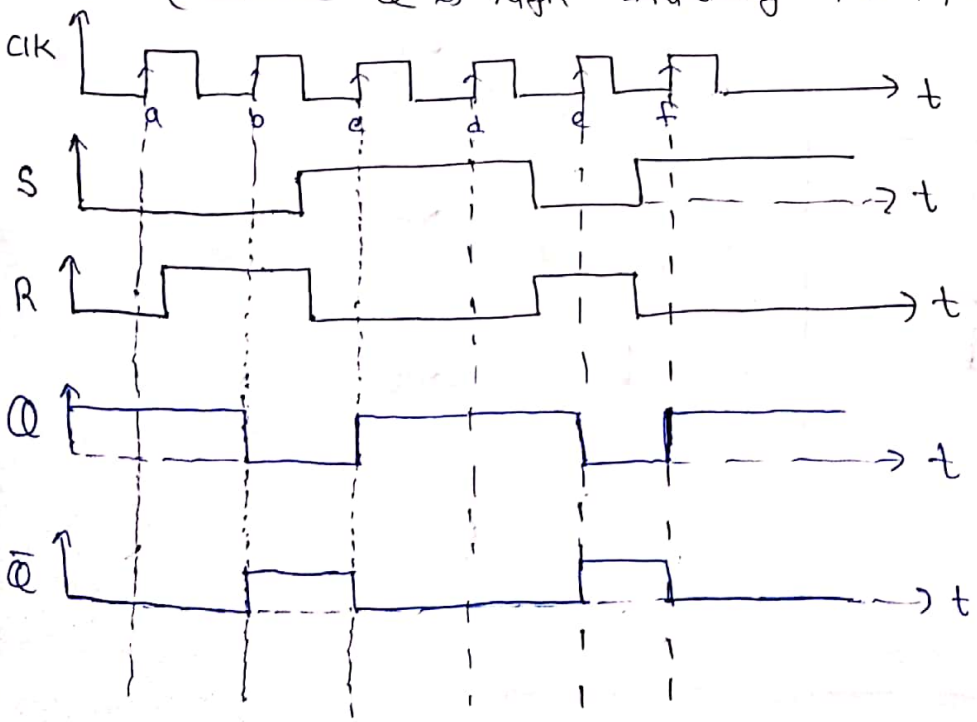
TRUTH TABLE :-

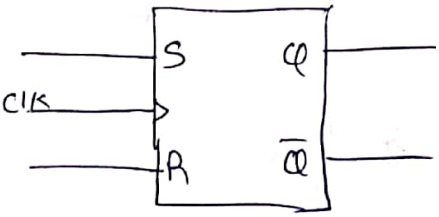
Inputs			Outputs
S	R	Clock	Q_{n+1}
0	0	↑	Q_n (no change)
0	1	↑	0 (Reset)
1	0	↑	1 (Set)
1	1	↑	? (Invalid)

(2) -ve edge triggered SR :-



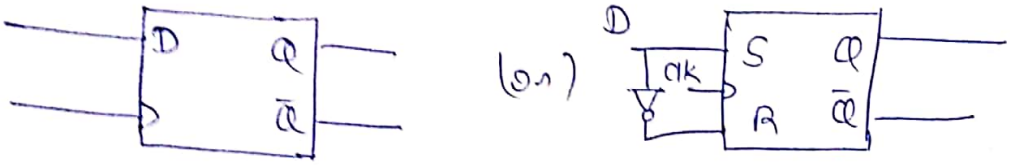
Eg:- Draw output waveform for the following input waveforms (Assume Q is high Initially i.e., Q = 1).





→ Edge triggered D flip-flop ←

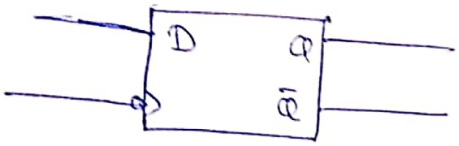
(1) +ve Edge triggering :-



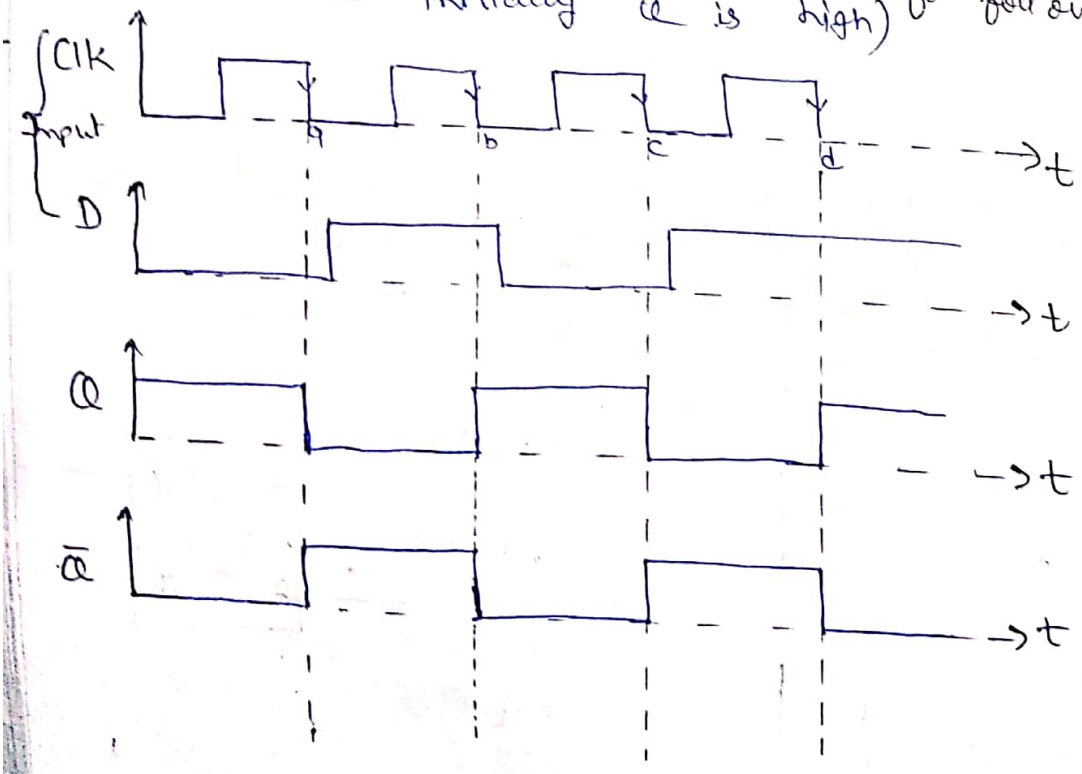
Truth - Table :-

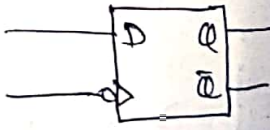
Input D	Clock	Output Q	
0	↑	0	(Reset)
1	↑	1	(set).

(2) -ve Edge triggering :-

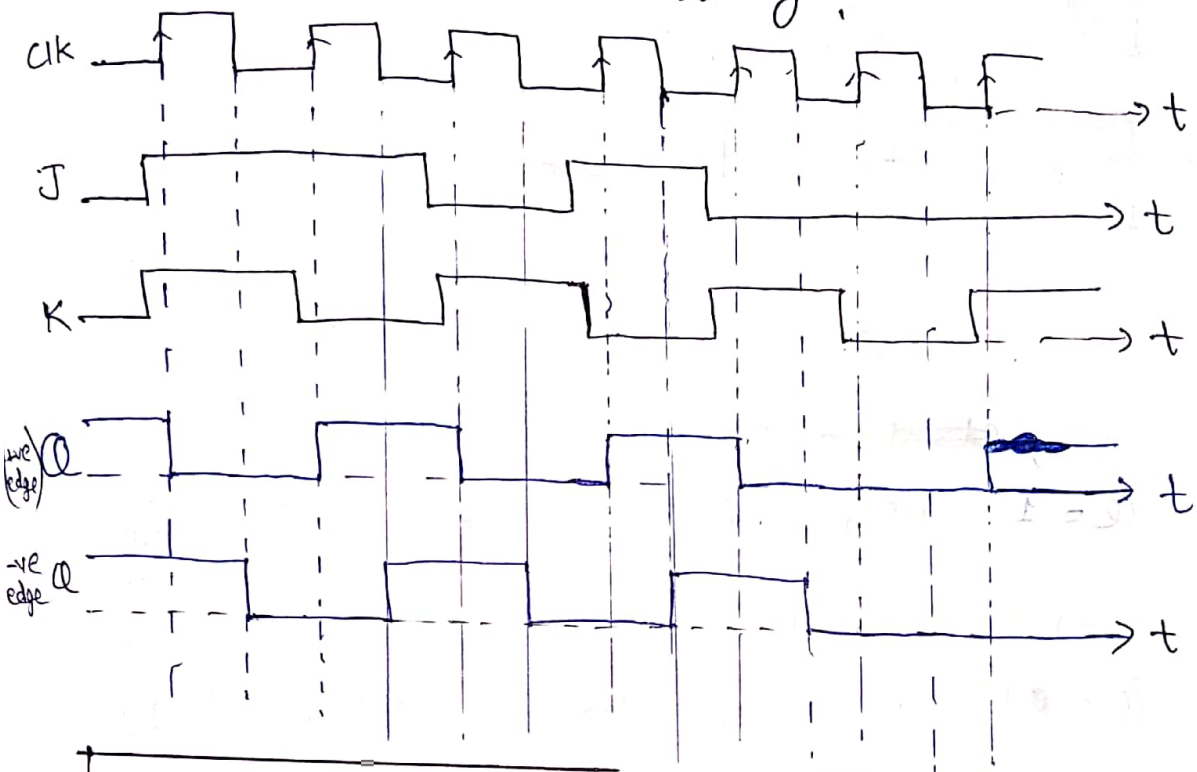


Eg:- Find the output waveform (Assume initially Q is high) for following input waveforms





Eg:- Draw output of JK for given input waveform.
(i) for +ve edge triggering,
(ii) for -ve edge triggering.



J	K	Clk	Q_{n+1}
0	0	↑	Q_n
0	1	↑	0
1	0	↑	1
1	1	↑	$\bar{Q}_n$

J	K	Clk	Q_{n+1}
0	0	↓	Q_n
0	1	↓	0
1	0	↓	1
1	1	↓	$\bar{Q}_n$

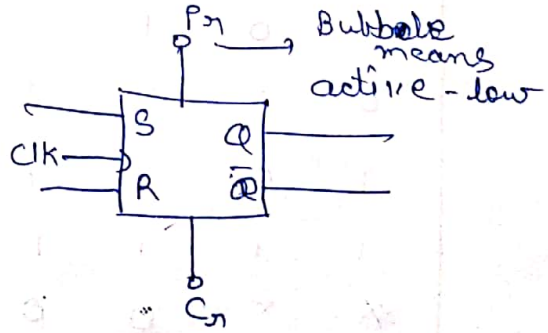
Flip-flops with asynchronous inputs (Preset (P_r), Clear (C_r)).

- Independent of clock inputs.

→ active low

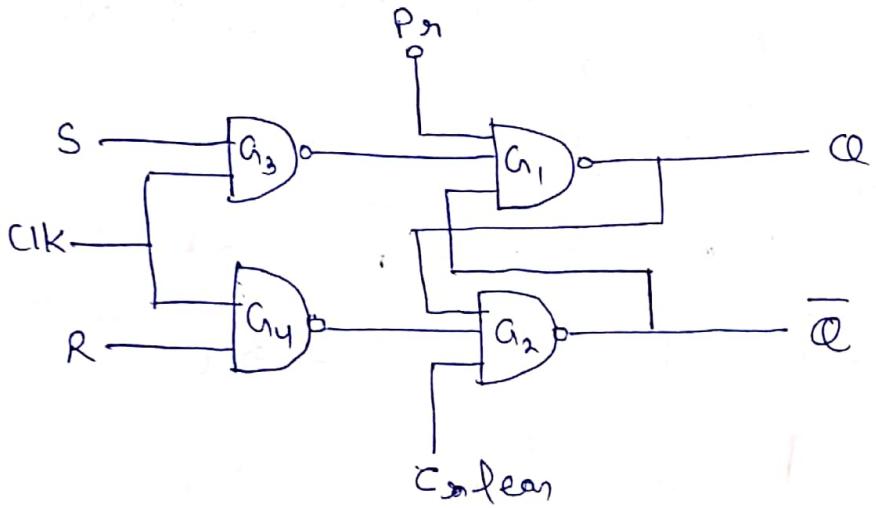
$\bar{P}_r$ $\bar{C}_r$

For Example,



Asynchronous SR flip-flop :-

→ Gate-level Diagram :-



(i) When $P_r = 0$, Clear = 1

$Q = 1$ i.e., flip-flop is set initially.

(ii) When $P_r = 1$ & Clear = 0

$\bar{Q} = 1$ i.e., flip-flop is reset initially.

(iii) When $P_r = 1$ & Clear = 1

Output depends on other inputs (S, R, clock).

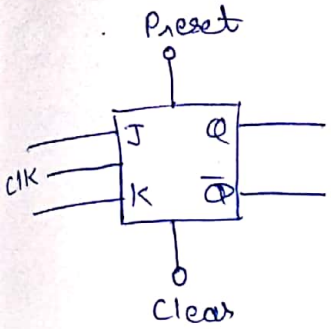
(iv) $\bar{P_r} = 0$ & Clear = 0

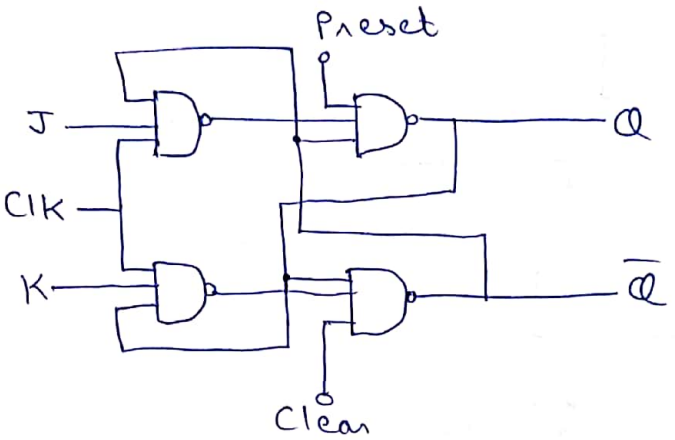
$Q = 1$, $\bar{Q} = 1$ ⇒ Not acceptable or permissible.

TRUTH TABLE :-

Inputs				Outputs
S_n	R_n	P_r	C_r	Q_{n+1}
x	x	0	1	1
x	x	1	0	0
0	0	1	1	Q_n
0	1	1	1	0
1	0	1	1	1
1	1	1	1	?
x	x	0	0	?

JK flip-flop with asynchronous inputs.





Inputs		Output
Pr	Cl	Q_{n+1}
0	0	not used
0	1	1 (Set)
1	0	0 (Reset)
1	1	Normal operation (depends on clock-pulse & inputs)

→ Truth-table will be same as SR with only one change when $P_r = C_r = S_n = R_n = 1 \Rightarrow Q_{n+1} = \bar{Q}_n$.

Flipo flop characteristic.

5 major Charact.
- → propogation delay (t_p)
- → Setup time (t_s)
- → Hold-up time (t_H)
- → Maximum clock frequency (f_{max}).
- → Asynchronous active pulse width.

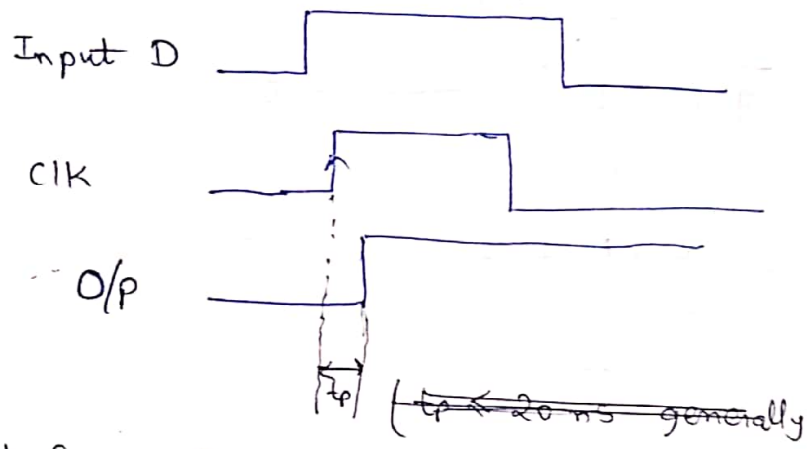
→ Clock-high pulse time.

→ Clock-low pulse time.

#(i) Propogation delay (t_p):-

→ It is the time difference/interval between the application of triggering edge of clock-pulse or asynchronous input & time at which the output actually makes a transition.

Eg :-



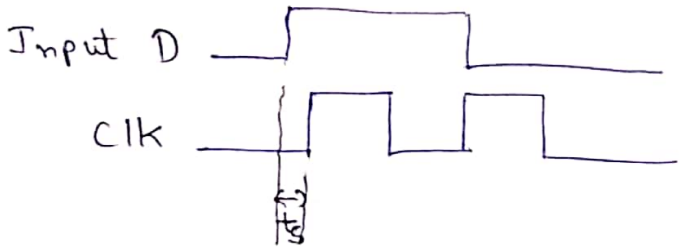
* Generally, t_p lies in (10 - 20 ns) i.e.,

$$10^{-8} \leq t_p \leq 2 \times 10^{-8} \text{ seconds.}$$

(2) Setup - time :-

→ It is the minimum amount of time for which the input must be maintained at a constant value before the clock ~~arrives~~ (edge) arrives.

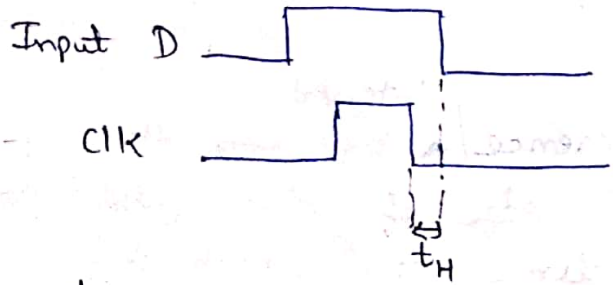
Eg :-



(3) Hold time (t_H) :-

→ It is the minimum amount of time for which input must be maintained at a constant value after the clock edge has been arrived.

Eg :-



* $10 \text{ ns} \leq t_H \leq 40 \text{ ns}$

4) $f_{max} = \frac{1}{t_s + t_p + t_{np}}$

t_{np} → Propagation delay of next-stage decoder

(5) Asynchronous active pulse width :-

→ The time required to set the initial condition after application of preset or clear.

ANALYSIS OF Clocked Sequential Circuits.

- What a given circuit will do under a certain operating conditions.
- Obtained from present-state inputs, outputs & flip-flop-states.
- Analysis starts from circuit diagram & culminates in a state table/diagram.
- Design :- Starts from set of specifications & generates logic diagram.

→ Behaviour of clocked Sequential circuits can be described by :-

→ State Equation
→ State Table
→ State Diagram
→ Flip-Flop equations.

State Equation

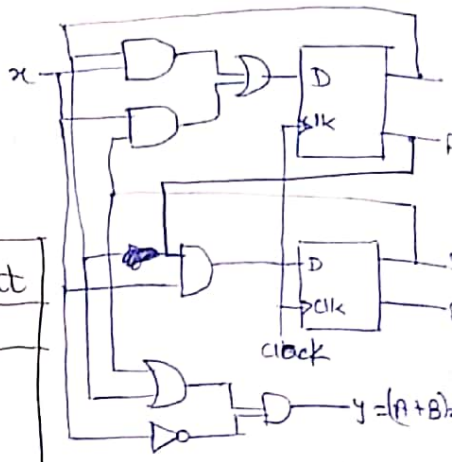
- Also called transition equation.
- L.H.S. denotes next-state & R.H.S. denotes present state.
- It is : next-state as a function of present-state & inputs.

→ State Table :-
(Transition Table)

- Consists of 4 sections : present state, input, next-state, output.

Eg :- $A(t+1) = Ax + Bx$

$B(t+1) = A'x$



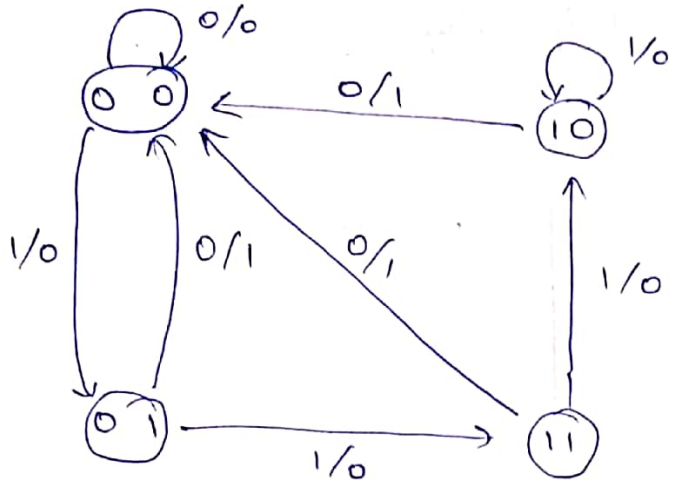
First Form :- $y = (A+B)x'$

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

2nd Form :-

Present State		Next State				Output	
		Input x = 0		Input x = 1		x = 0	x = 1
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

→ State - Diagram :-



Note :- If no output is there & input is single, then it will be denote by x. (No '/' will be used)

→ Flip-Flop Equations :-

• Output Equations :-

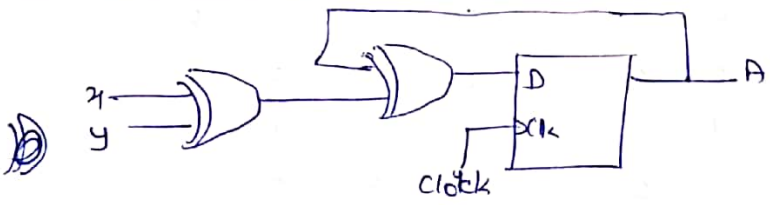
• Input Equations (Excitation Equations) :-

Eg :- Input Eqn :- $D_A = Ax + Bx$

$D_B = A'x$.

Output Eqn :- $y = (A+B)x'$

Eg :-

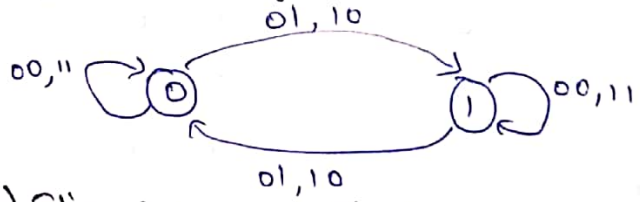


(I) State Eqn :- $A(t+1) = A \oplus x \oplus y$.

(II) State Table :-

Present State A	Inputs x	Inputs y	Next State A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(III) State Diagram :-

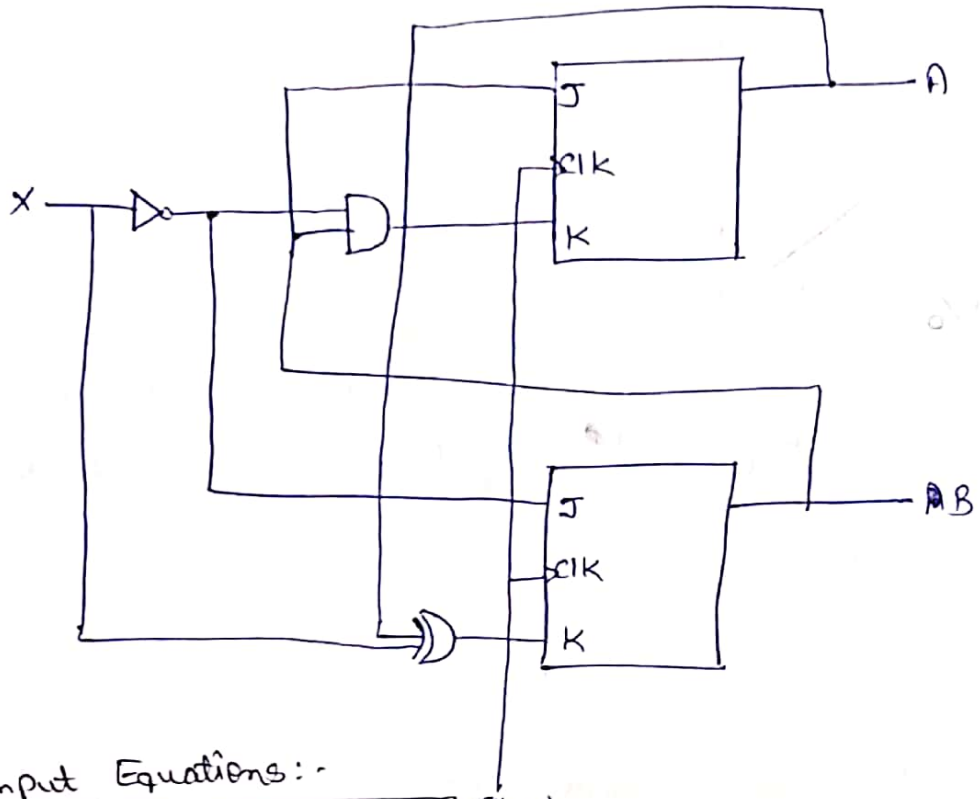


(IV) Flip-Flop Equations :

Input Eqn. :- $D_A = A \oplus x \oplus y$.

$KA + J$.

Eg:- Analysis of JK Flip-flops.



Input Equations:-

$J_A = B$, $K_A = Bx'$

$J_B = x'$, $K_B = A \oplus x$

State Table:-

Present State		Input	Next State		Flip-Flop Inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

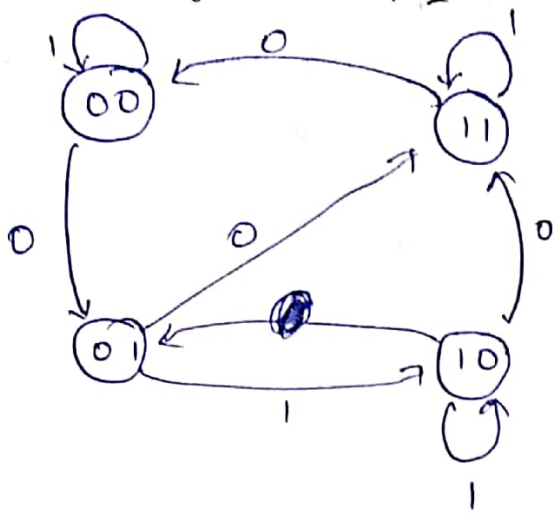
Using Characteristic table:-

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

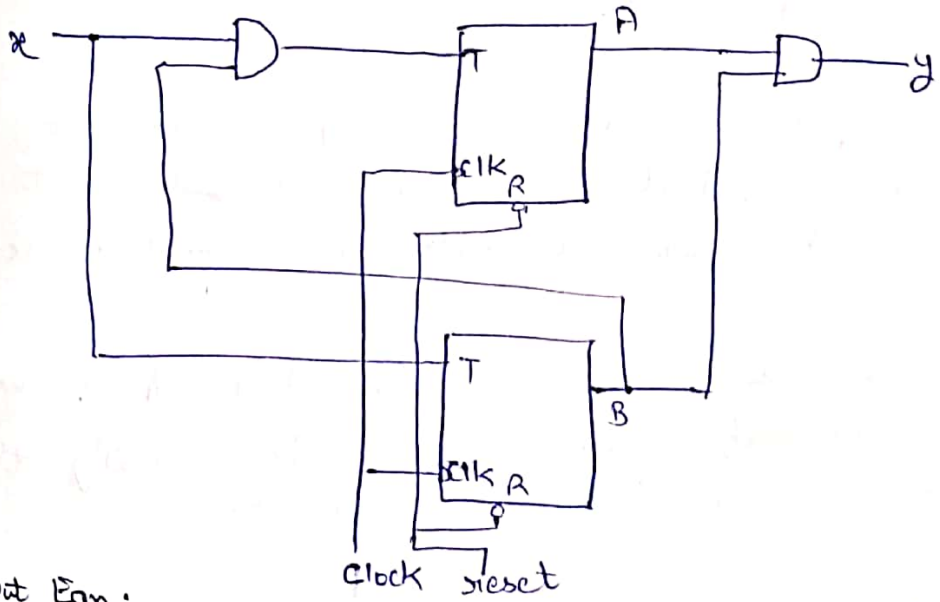
Characteristic Eqn:-

$Q(t+1) = JQ' + K'Q$.

State - Diagram :-



Eg:- Analysis of Clocked Sequential Circuits with T flip-flops.



Input Eqn :-

$T_A = Bx$, $T_B = x$.

Output Eqn : $y = AB$

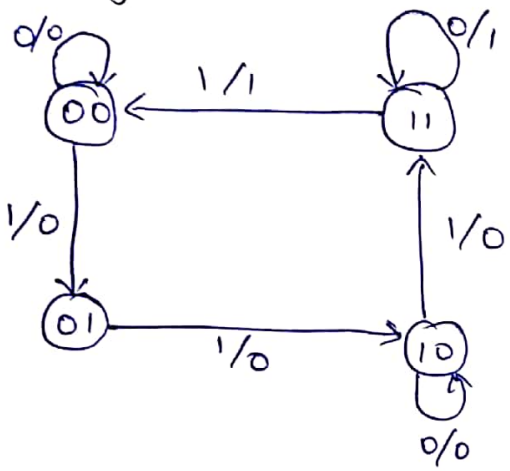
State - Table :-

Present A	Present B	Input x	Next A	Next B	Output y	T_A	T_B
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1
0	1	0	0	1	0	0	0
0	1	1	1	0	0	1	1
1	0	0	1	0	0	0	0
1	0	1	1	1	0	0	1
1	1	0	1	1	1	0	0
1	1	1	0	0	1	1	1

State Eqn :- $A(t+1) = (Bx)'A + (Bx)A' = (Bx) \oplus A$

$B(t+1) = x'B + xB' = x \oplus B$

State- Diagram :-



Flip-Flop Equations :-

$T_A = Bx$

$T_B = x$

$y = AB$

State Reduction & Assignment

→ Two sequential circuits may exhibit the same input-output behaviour, but have a different no. of internal states (flip-flops) in their state diagram.

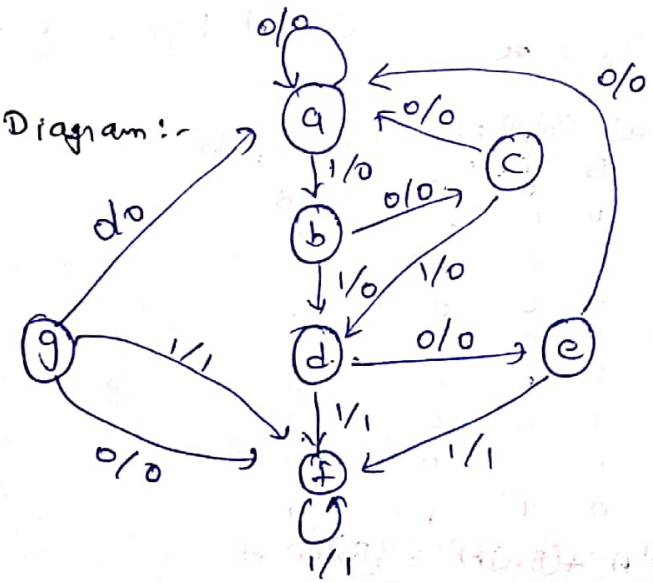
→ Reducing the no. of states

→ If identical input sequences are applied to two circuits & identical outputs occur for all input sequences; then the two circuits are said to be equivalent.

→ Two states are said to be equivalent if for each member of set of inputs, they give exactly the same outputs & send the circuit either

Eg :-

State-Diagram :-



State - Table :-

Present-state	Next-State		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e →	a	f	0	1
f	g	f	0	1
g →	a	f	0	1

* Present - state e & g are equivalent.

⇓

Present - state	Next - state		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d →	e	f	0	1
e	a	f	0	1
f →	e	f	0	1

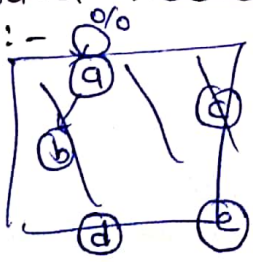
* Now, d & f states are equivalent.

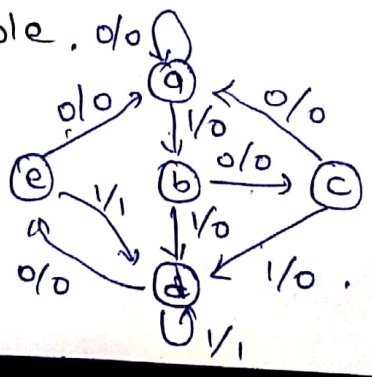
Present - state	Next - state		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

* Now, No more equivalent states.

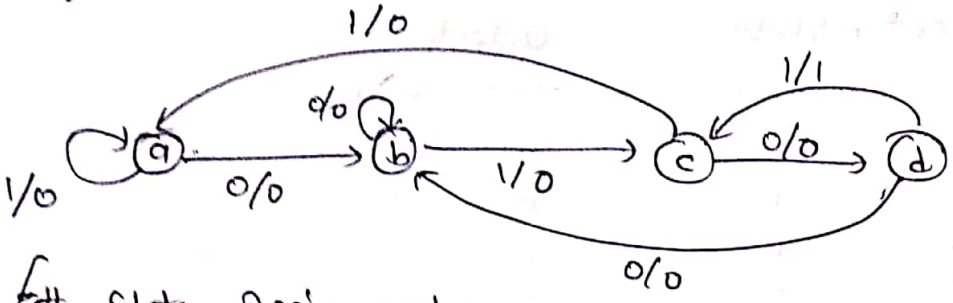
Reduced So, No further reduction is possible.

State - Digram :-





Eg :-



State Assignment :-

→ In order to design a sequential circuit with physical components, it is necessary to assign unique coded binary values to the states.

→ For a circuit with m states, the codes must contain n bits, where $m \leq 2^n$.

→ Unused states are treated as Don't cares.

Eg :- State - Table :-

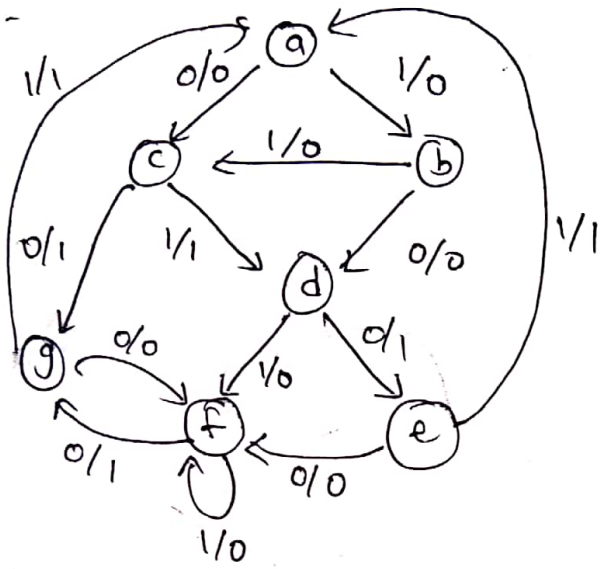
Present - State	Next - State		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
a	b	a	0	0
b	b	c	0	0
c	d	a	0	0
d	b	c	0	1

→ No reduction is possible here.

State - Assignment :-

Variable	Assigned Value
a	00
b	01
c	10
d	11

Eg:-



State- Table:-

Present-State	Next-State		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
a	c	b	0	0
b	d	c	0	1
c	g	d	1	1
d	e	f	1	0
e →	f	a	0	1
f	g	f	1	0
g →	f	a	0	1

⇓

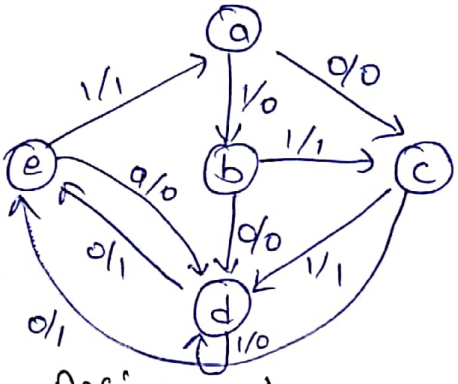
a	c	b	0	0
b	d	c	0	1
c	e	d	1	1
d →	e	f	1	0
e	f	a	0	1
f →	e	f	1	0

⇓

a	c	b	0	0
b	d	c	0	1
c	e	d	1	1
d	e	d	1	0
e	d	a	0	1

No reduction possible now.

State - Diagram :-



State - Assignment :-

Variable	Assigned Value
a	000
b	001
c	010
d	011
e	100

Design Procedure :-

Procedure for designing synchronous sequential circuits is as follows:-

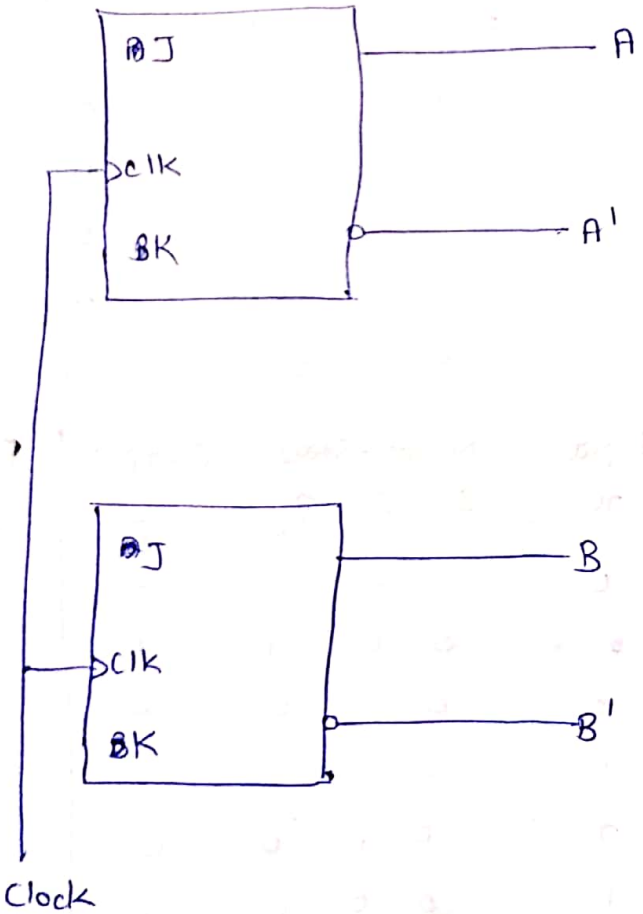
(1) Derive a state - diagram
(2) Reduce no. of states if necessary
(3) Assign binary values to the states.
(4) Obtain the binary - coded state table (Transition Table)
(5) Choose type of flip-flops to be used,
(6) Derive simplified flip-flop input equations & output equations.
(7) Draw logic diagram.

Synthesis Using JK - flip-flops

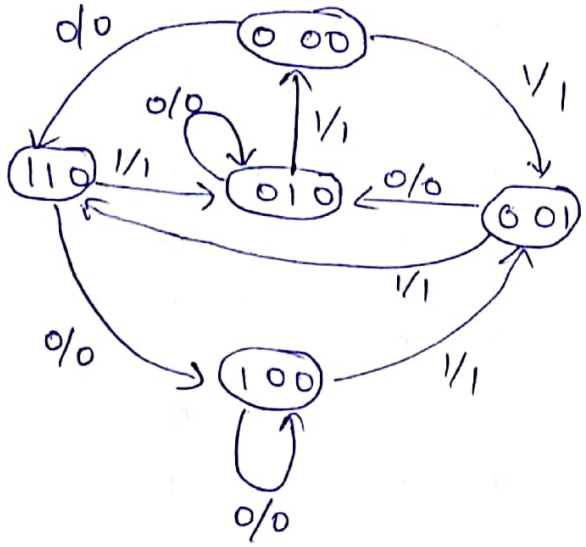
Eg:-

Present State		Input	Next State		Flip-flop inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Using K-Map : $J_A = Bx'$, $K_A = Bx$, $J_B = x$, $K_B = (A \oplus x)'$



Que:- Using D -flip-flop, design a clocked sequential circuit whose state-diagram is given below.



State - Table:-

Present - state	Next - State		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
000	110	001	0	1
001	010	110	0	1
010	010	000	0	1
100	100	001	0	1
110	100	010	0	1

Present - State			Input	Next - State			Output	Flip - Flop Inputs		
Q_2	Q_1	Q_0	x	Q_2	Q_1	Q_0	y	D_2	D_1	D_0
0	0	0	0	1	1	0	0	1	1	0
0	0	0	1	0	0	1	1	0	0	1
0	0	1	0	0	1	0	0	0	1	0
0	0	1	1	1	1	0	1	1	1	0
0	1	0	0	0	1	0	0	0	1	0
0	1	0	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0	1	0	0
1	0	0	1	0	0	1	1	0	0	1
1	1	0	0	1	0	0	0	1	0	0
1	1	0	1	0	1	0	1	0	1	0

By K-Map :-

For D_2 :-

Q_2Q_1 \ Q_0x	00	01	11	10
00	1	0	1	0
01	0	0	X	X
11	1	0	X	X
10	1	0	X	X

$$D_2 = Q_0x + Q_2\bar{x} + \bar{Q}_1\bar{Q}_0\bar{x}$$

For D_1 :-

Q_2Q_1 \ Q_0x	00	01	11	10
00	1	0	1	1
01	1	0	X	X
11	0	1	X	X
10	0	0	X	X

$$D_1 = Q_0 + \bar{Q}_2\bar{x} + Q_2Q_1x$$

For D_0 :-

Q_2Q_1 \ Q_0x	00	01	11	10
00	0	1	0	0
01	0	0	X	X
11	0	0	X	X
10	0	1	X	X

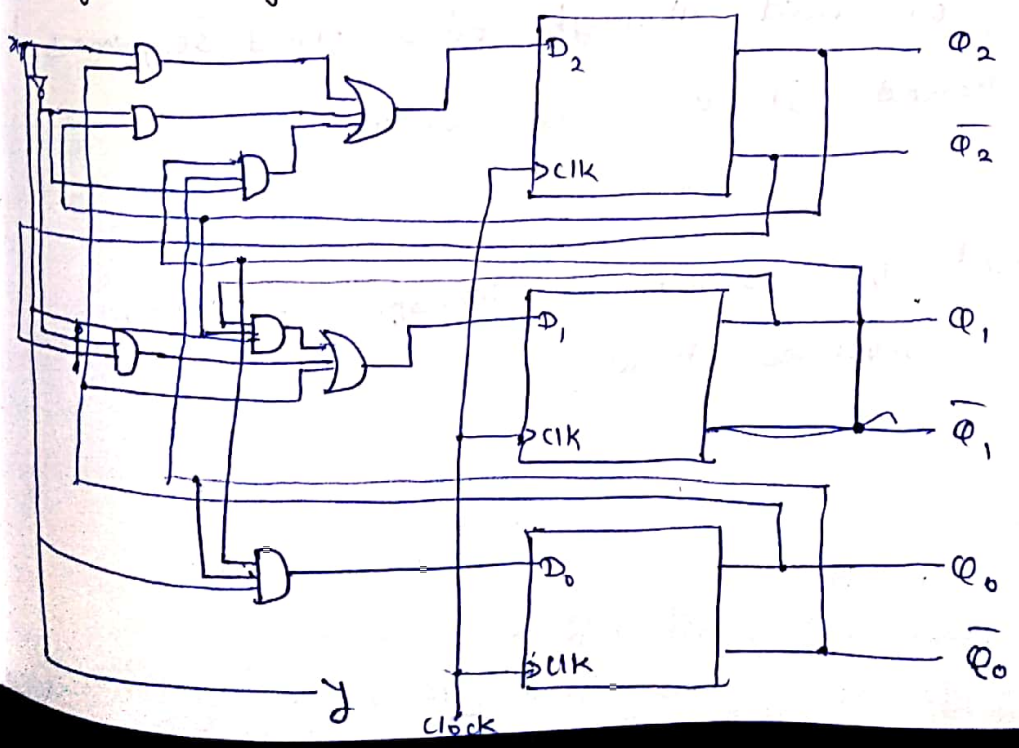
$$\Rightarrow D_0 = \bar{Q}_1\bar{Q}_0x$$

For y :-

Q_2Q_1 \ Q_0x	00	01	11	10
00	0	1	1	0
01	0	1	X	X
11	0	1	X	X
10	0	1	X	X

$$\Rightarrow y = [redacted]x$$

Logic - Diagram :-



Application of Flip-Flops

- Flip-Flop is the minimum unit of storage,
  - → Registers (Buffer or stores data)
  - → Counters (Counting ~~from 0 to 9~~ of predefined sequence repeated)
  - → Shift Registers: holds group of data which can be stored & shifted (transfer).

Registers:-

→ A register is a memory device which stores more than one bit of information,

→ It is a generalization of a flip-flop.
  * A Flip-Flop is a 1-bit register.

→ Main Operations:-

(i) Load or Store :- Put new data into the register.

(ii) Read :- Retrieve the data without altering it.

Counters

→ Specific type of sequential circuits or registers.

→ They are sequential circuits which "count" through a specific & pre-defined state sequence. They can count up, count down, or count through other fixed sequences;

→ If not mentioned, it is a up counter.

Shift Registers

→ They are used for storage & transfer of data,

→ No specific sequence here.

Benefits of counters :

- They can act as simple clocks :-
  - → How many bits have been sent or received?
  - → How many steps have been performed in some computation?
- All processors contain a program counter (PC).
- In computers, they are used to increment or decrement by one in response to input.
- A counter that follows the binary number sequence is called a binary counter,
  - → n bit binary counter : n flip-flops, counts in binary from 0 to 2^{n-1}.

Counter used in Modulus :-

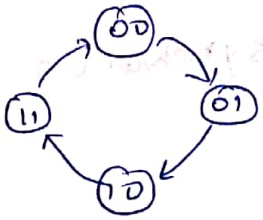
- Each count is called state of counter
- A counter with n flip-flops can have 2^n states & of same modulus.
- The no. of states in a counter is known as its mod number.
- Thus, a 2 bit counter is a mod-4 counter,
- A mod-n counter may also be described as a divide-by-n counter.

Eg :-

2 bit counter ≡ mod-4 counter ≡ divide-by-4 counter,

- Counter is a clocked sequential circuit whose state diagram contains a single cycle,

Eg :-

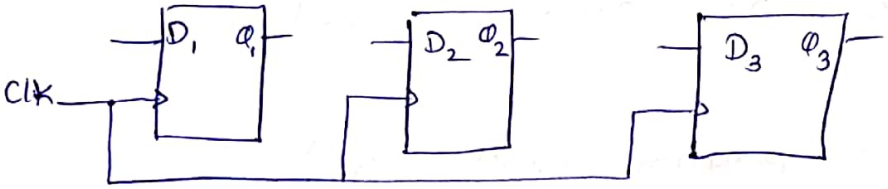


- Eg :- mod-12 counter counts from 0 (0000) to 11 (1011) & requires 4 flip-flops (16 states - only 12 used),
- Thus a counter with non-Power of 2 modulus has unused states.

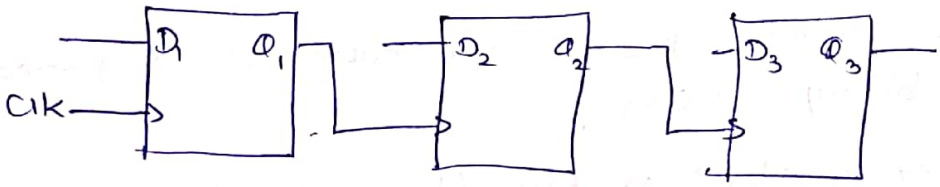
Types of Counter:-

- Synchronous Counters (Parallel)
- Asynchronous Counters (Ripple counters)

→ Synchronous counter:-

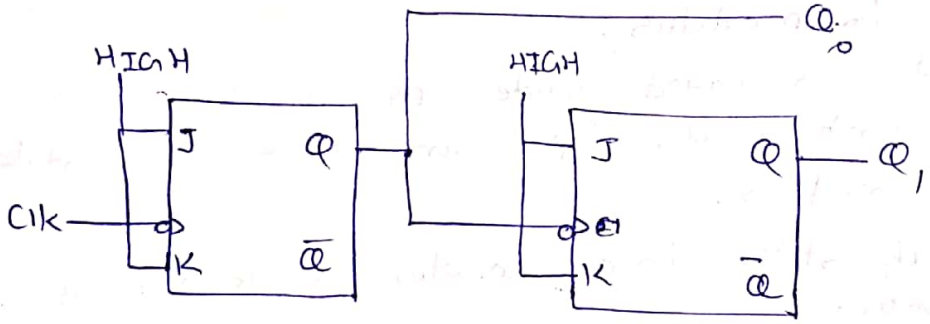


→ Asynchronous counter:-

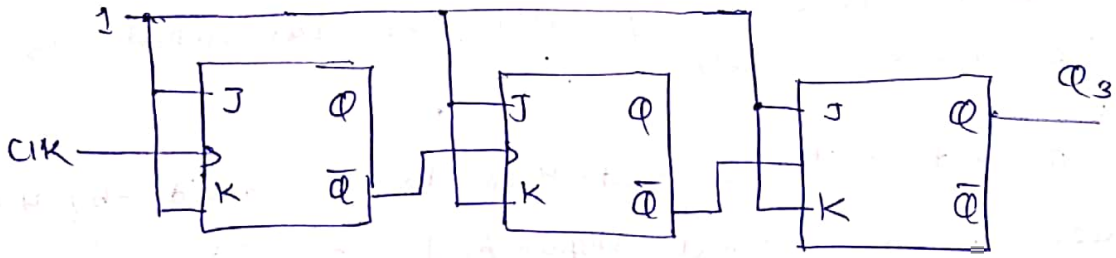


⇒ Huge Propogation Delay
Complex Design.

Eg:- (a) 2 bit Asynchronous counter:- (0 to 3)



(b) 3 bit binary down-counter:-



Synchronous Counter (Parallel)

→ Eg:- Let's design 3-bit synchronous Counter using JK flip-flop.

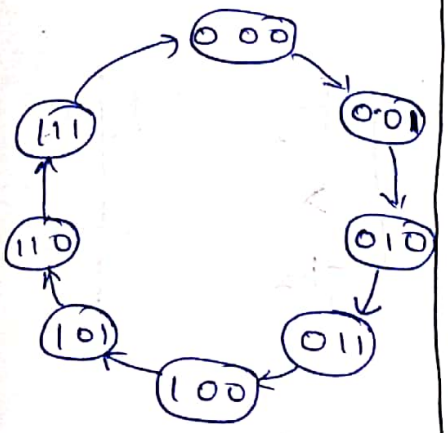
Solution:-

Counter = mod-8 counter.

Total no. of flip flops = 3.

→ State- Diagram :-

State - Table.

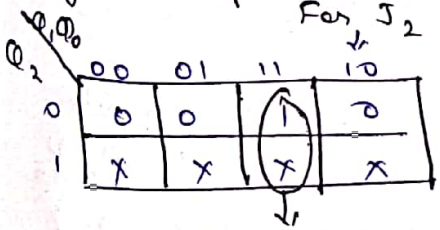


→ State Table :-

Present - state			Next - state			F/F Inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

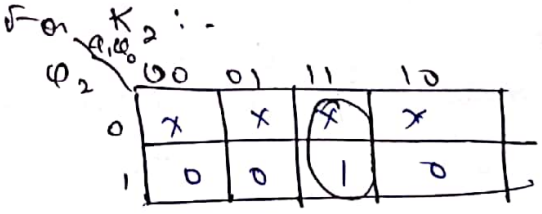
Using K-map :-

For J_2



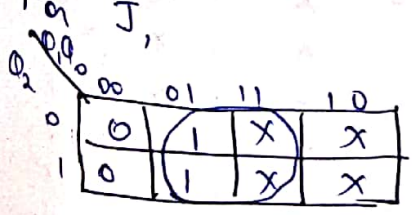
$J_2 = Q_1 Q_0$

For K_2 :-



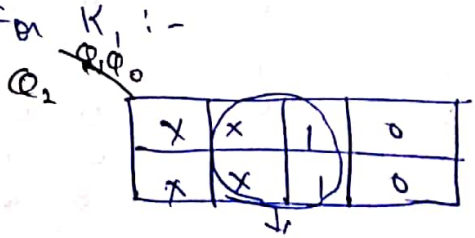
$K_2 = Q_1 Q_0$

For J_1



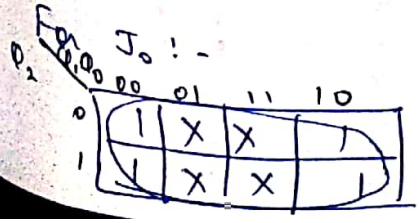
$J_1 = Q_0$

For K_1 :-



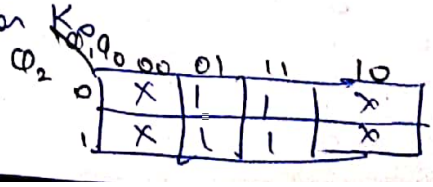
$K_1 = Q_0$

For J_0 :-



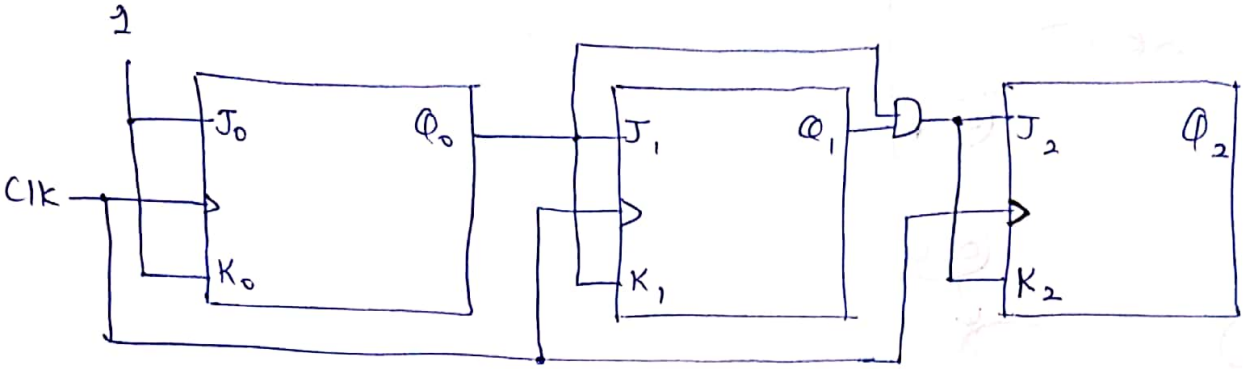
$J_0 = 1$.

For K_0



$\Rightarrow K_0 = 1$

Logic - Diagram :-

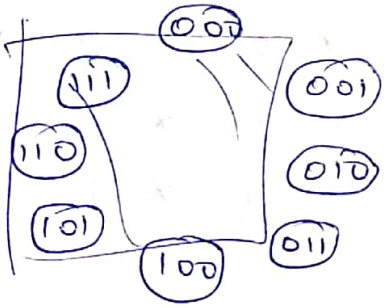


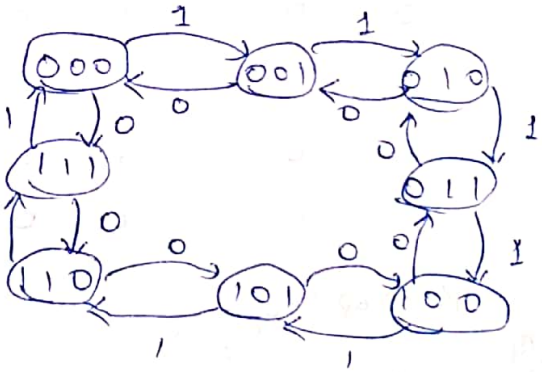
Eg 2:- Design a 3 bit UP/DOWN (bidirectional) counter using JK flip flops.

Method 1:- Design both up & down counters separately & then combine them accordingly.

Method 2:- Take a control input : M = up / down

State - Diagram :-





State - Table :-

Present - State			Control Input	Next - State			Flip - Flop Inputs					
Q_2	Q_1	Q_0	M	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	1	1	1	1	x	1	x	1	x
0	0	0	1	0	0	1	0	x	0	x	1	x
0	0	1	0	0	0	0	1	x	1	x	x	1
0	0	1	1	0	1	0	0	x	1	x	x	1
0	1	0	0	0	0	1	1	x	x	1	1	x
0	1	0	1	0	1	1	0	x	x	0	1	x
0	1	1	0	0	1	0	1	x	x	1	x	1
0	1	1	1	1	0	0	1	x	x	1	x	1
1	0	0	0	1	0	1	x	1	1	x	1	x
1	0	0	1	0	1	1	x	0	0	x	1	x
1	0	1	0	1	0	0	x	1	1	x	x	1
1	0	1	1	0	1	0	x	0	0	x	x	1
1	1	0	0	1	0	0	x	0	x	1	0	x
1	1	0	1	1	0	1	x	0	x	1	1	x
1	1	1	0	1	1	0	x	0	x	0	x	1
1	1	1	1	1	1	0	x	0	x	0	x	1

K- Map.

For J_2 :

Q_2Q_1 \ Q_0M	00	01	11	10
00	1	0	0	0
01	0	0	1	0
11	X	X	X	X
10	X	X	X	X

$J_2 = Q_0 M Q_1 + \bar{Q}_0 \bar{M} \bar{Q}_1$

For K_2 :

Q_2Q_1 \ Q_0M	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	0	0	1	0
10	1	0	0	0

$K_2 = Q_0 M Q_1 + \bar{Q}_0 \bar{M} \bar{Q}_1$

Design of synchronous counters with unused states:-

•) self - starting counter :-

→ whenever counter goes to an invalid state, it comes back to a valid state in 1 or 2 clock pulses & then starts counting in a normal sequence, it is called self starting.

•) Lock out condition:-

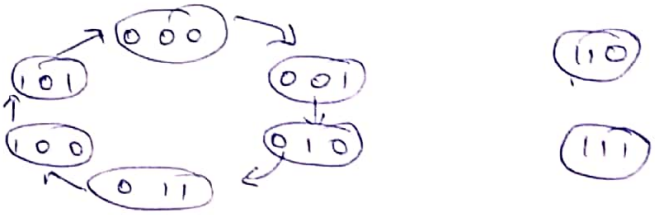
→ If a counter an invalid state, it keeps on moving from one invalid state to another when clock pulses are applied.

→ It never goes returns to a valid state.

Eg:-(1) Design a mod-6 synchronous counter using J|K flip flops, check whether the counter is self starting.

Solution:-

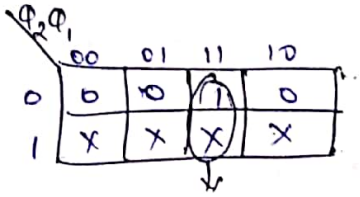
State- Diagram:-



state - Diagram:-

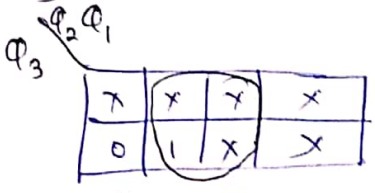
Present - state			Next - State			Inputs to flip-flops					
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1

For J_3



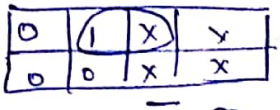
$J_3 = Q_2 Q_1$

For K_3



$K_3 = Q_1$

For J_2

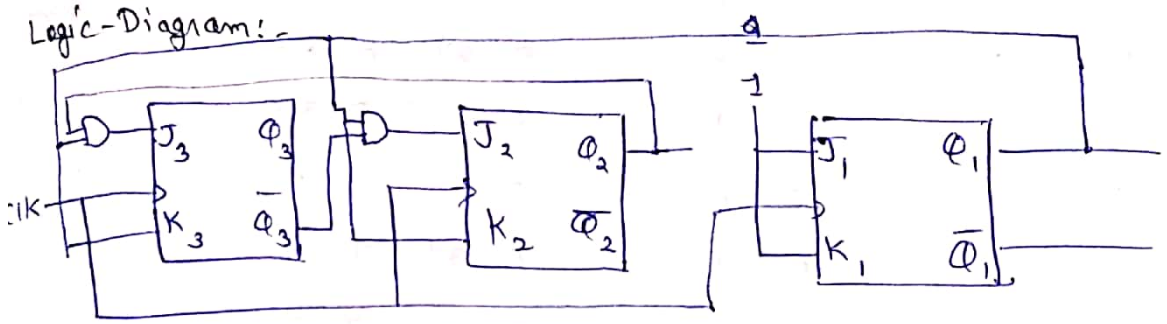


$J_2 = \overline{Q}_3 Q_1$

$K_2 = Q_1$

$J_1 = Q_3 \cdot 1$ $\qquad$ $K_1 = 1$,

Logic-Diagram:-



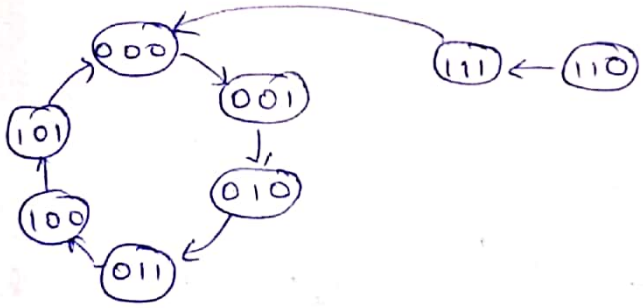
Let's Check

Transition Table:-

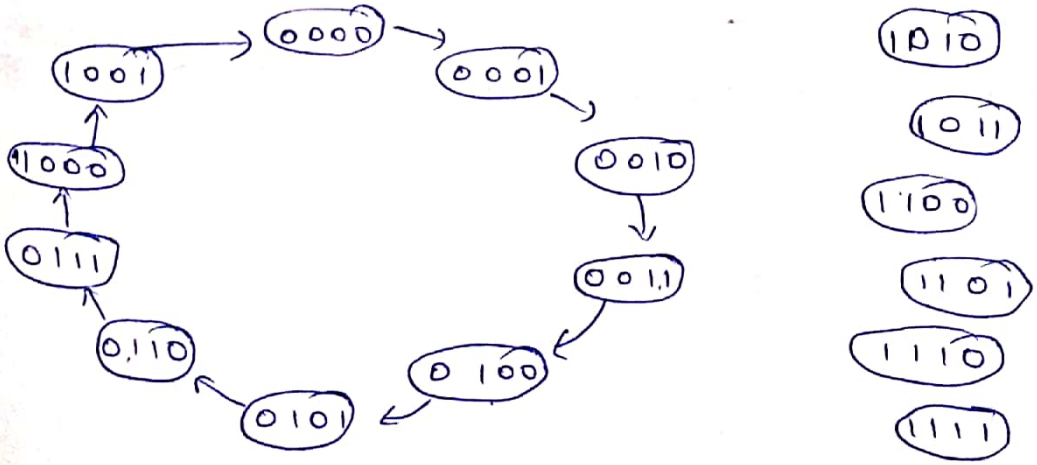
Present - State			F/F Inputs						Next - State.		
Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1	Q_3	Q_2	Q_1
1	1	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	0	1	1	1	0	0	0

Thus, The counter is self-starting as it returns back to valid state in 2 clock-pulses.

State-Diagram:-



Eg:- Design a BCD-counter using JK-flip flops. Check whether the counter is self starting.



State-Table:-

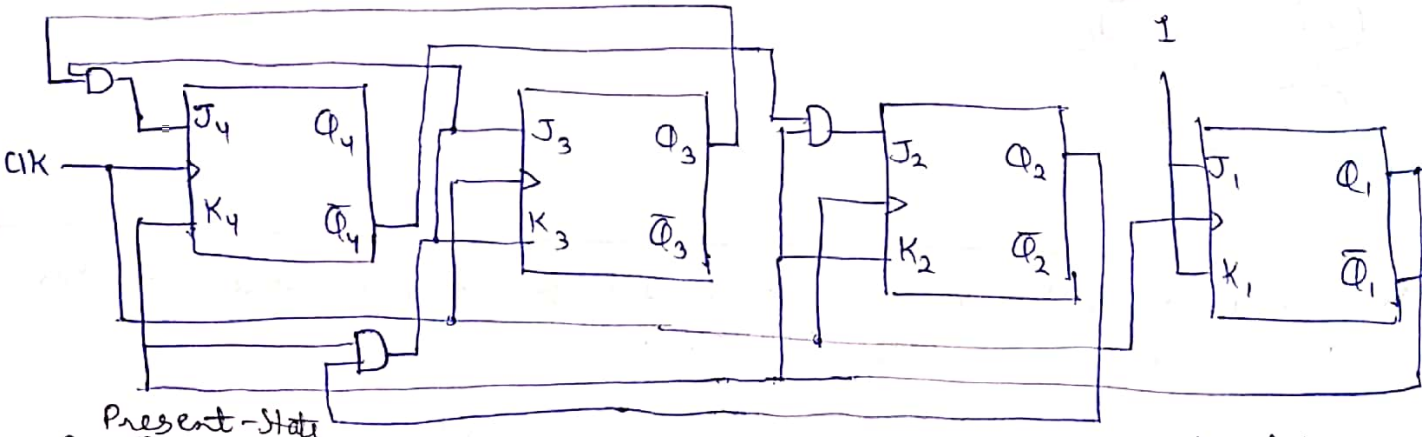
Present-State				Next-State				F/F Inputs							
Q_4	Q_3	Q_2	Q_1	Q_4	Q_3	Q_2	Q_1	J_4	K_4	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	0	0	0	0	X	1	0	X	0	X	1	X

$J_4 = Q_1 Q_2 Q_3$, $K_4 = Q_1$

$J_3 = Q_1 Q_2 = K_3$

$J_2 = \overline{Q_4} Q_1$, $K_2 = Q_1$

$J_1 = K_1 = 1$



Present-State $Q_4\ Q_3\ Q_2\ Q_1$ $J_4\ K_4\ J_3\ K_3\ J_2\ K_2\ J_1\ K_1$ Next-state, $Q_4\ Q_3\ Q_2\ Q_1$

0 0 X
0 1 X
1 0 X
1 1 X

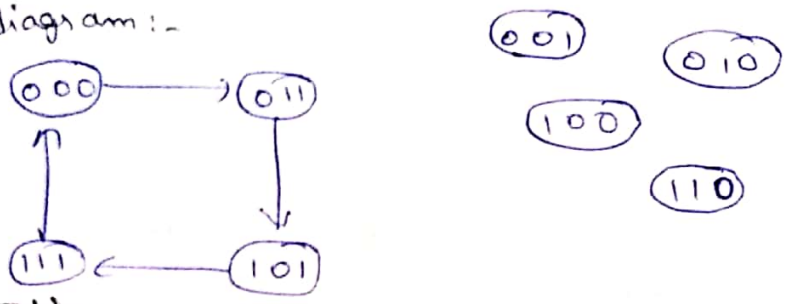
*Assignment :- Produce 3-bit gray code counter.

~~State D~~ i'

Eg:(3) Using T flip-flop, design a synchronous counter through states 0, 3, 5, 7, 0 . . .

Soln :-

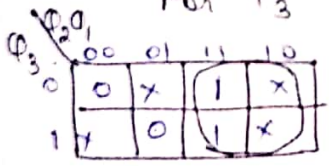
State-diagram :-



State-Table :-

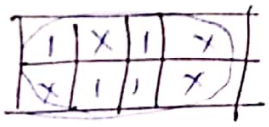
Present-State			Next-State			F/F Inputs		
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	T_3	T_2	T_1
0	0	0	0	1	1	0	1	1
0	1	1	1	0	1	1	1	0
1	0	1	1	1	1	0	1	0
1	1	1	0	0	0	1	1	1

For T_3



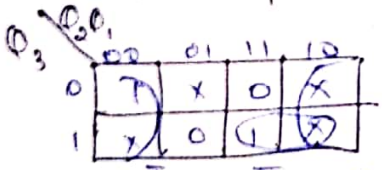
$T_3 = Q_2$

For T_2 :-



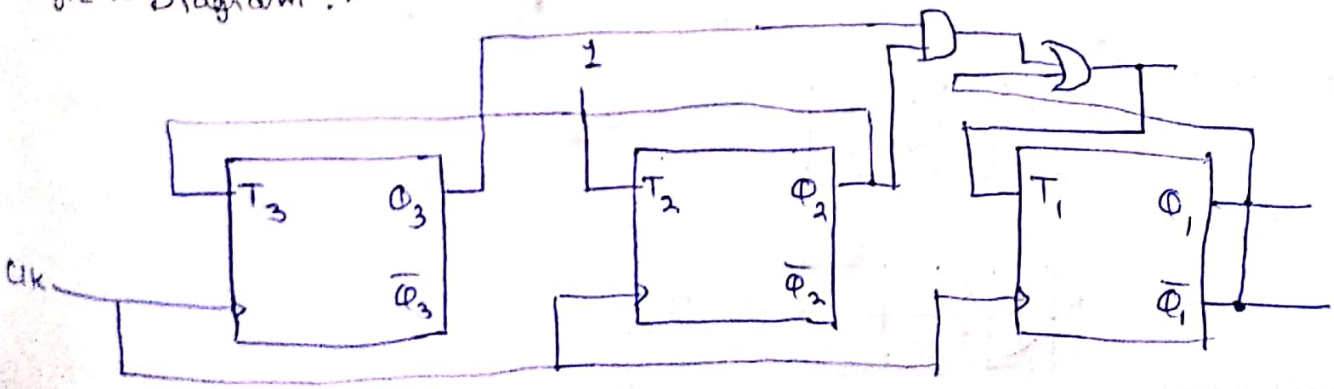
$T_2 = 1$

For T_1 :-



$T_1 = \bar{Q}_1 + Q_2 Q_3$

Logic-Diagram :-



Let's check if it is self-starting or not.

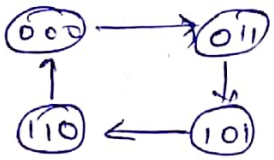
Transition-Table:-

Present-State			F/F Inputs			Next-State		
Q_3	Q_2	Q_1	T_3	T_2	T_1	Q_3	Q_2	Q_1
0	0	1	0	1	0	0	1	1
0	1	0	1	1	1	1	0	1
1	0	0	0	1	1	1	1	1
1	1	0	1	1	0	0	0	1

Since, each invalid state returns to valid-state in less than or equal to two clock-pulses,

Hence, It is self-starting synchronous counter.

Eg:- States: 0, 3, 5, 6, 0, ...



State-Table:-

Present-State			Next			F/F Inputs		
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	T_3	T_2	T_1
0	0	0	0	1	1	0	1	1
0	1	1	1	0	1	1	1	0
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0

For T_1 :-

Q_3 \ Q_2Q_1	00	01	11	10
0	1	X	0	X
1	X	X	X	0

$T_1 = \overline{Q_2}$

For T_3 :-

Q_3 \ Q_2Q_1	00	01	11	10
0	0	X	1	X
1	X	0	X	1

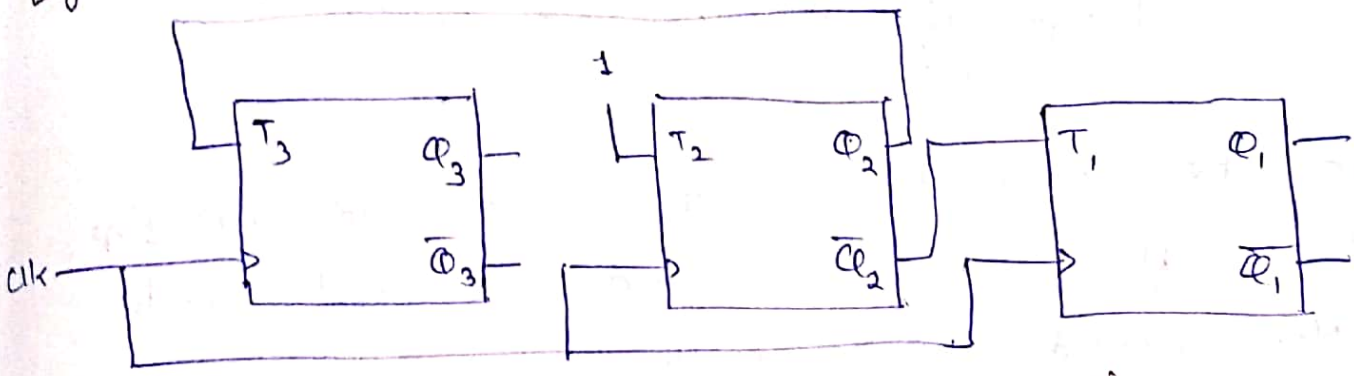
$T_3 = Q_2$

For T_2 :-

	00	01	11	10
0	1	X	1	X
1	X	1	X	1

$T_2 = 1$

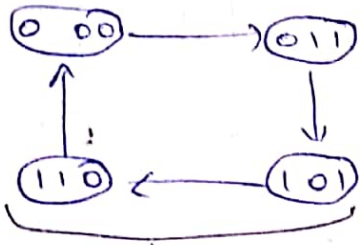
Logic - Diagram :-



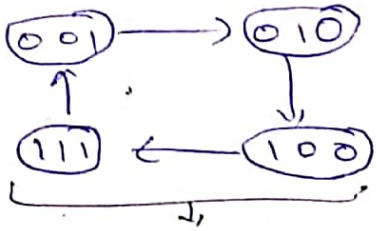
Now, let's check now if it is self starting :-

Present			FF Inputs			Next - State		
Q_3	Q_2	Q_1	T_3	T_2	T_1	Q_3	Q_2	Q_1
0	0	1	0	1	1	0	1	0
0	1	0	1	1	0	1	0	0
1	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	1

Thus, State - Diagram :- ✱ It is not self - starting & in lock - out condition.



Valid - States



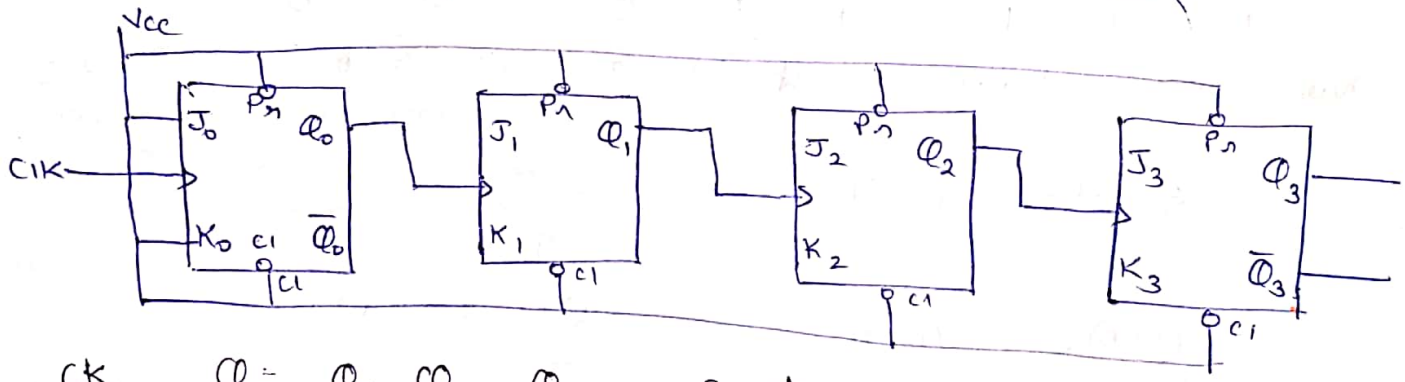
Invalid & unused states.

✱ Assignment :- How to eliminate lock - out condition & rectify it.

Design of Asynchronous Counter
(Serial Counter)

- Can be designed using T or JK flip-flop.
- These flip-flops are used in toggle mode.
- Not clocked simultaneously.
- Output of previous flip-flop becomes input to next flip-flop, hence asynchronous.
- Inputs J & K are 1, this acts as toggle switch & output of flip-flop is complemented.

Eg:- 4 bit ripple counter :-

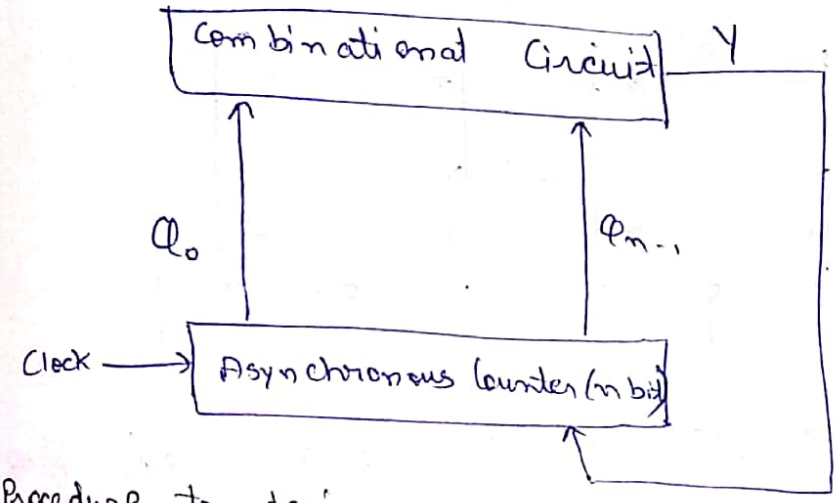


CK	Q_3	Q_2	Q_1	Q_0	Count
0	0	0	0	0	0
1	0	0	0	1	1
15	1	1	1	1	15

Mod n Asynchronous Counter

- n bit asynchronous counter counts $N = 2^n$ clock pulses.
- Reset terminal is used.
- A combinational circuit is designed such that all the flip-flops are reset after count n.

Block - diagram :-



Procedure to design :-

- Min. no. of flip-flops required ($n : M < 2^n$).
- Prepare the sequence.
- Draw truth table with output of combinational circuit (Y) such that $Y = 1$ for valid states & $Y = 0$ for invalid states.

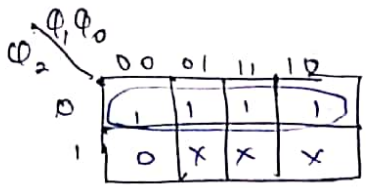
Eg:- Mod 4 ripple counter using JK flip flops.

(1) $M < 2^n$, $4 < 2^n \Rightarrow n = 3$

(2) Design a comb^n circuit with output Y such that all FFs are cleared after 4 clock pulses.

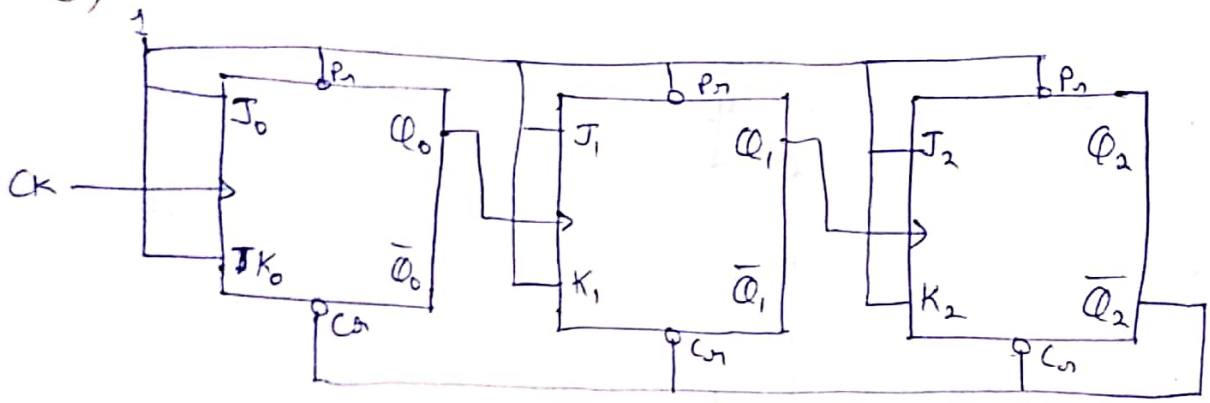
(3) $Y = 1$ for count 0–3 & $Y = 0$ for count = 4.

Clock	FF states Q_2	Q_1	Q_0	Output Y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	X
6	1	1	0	X
7	1	1	1	X



$Y = \overline{Q}_2$

(4)

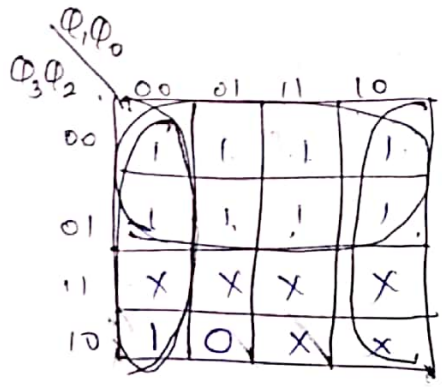


Design a mod 9 ripple counter using T flip-flops.

(1) $\because 9 < 2^n \Rightarrow n = 4$.

(2) $Y = 1$ for 0 - 8 & $Y = 0$ for 9.

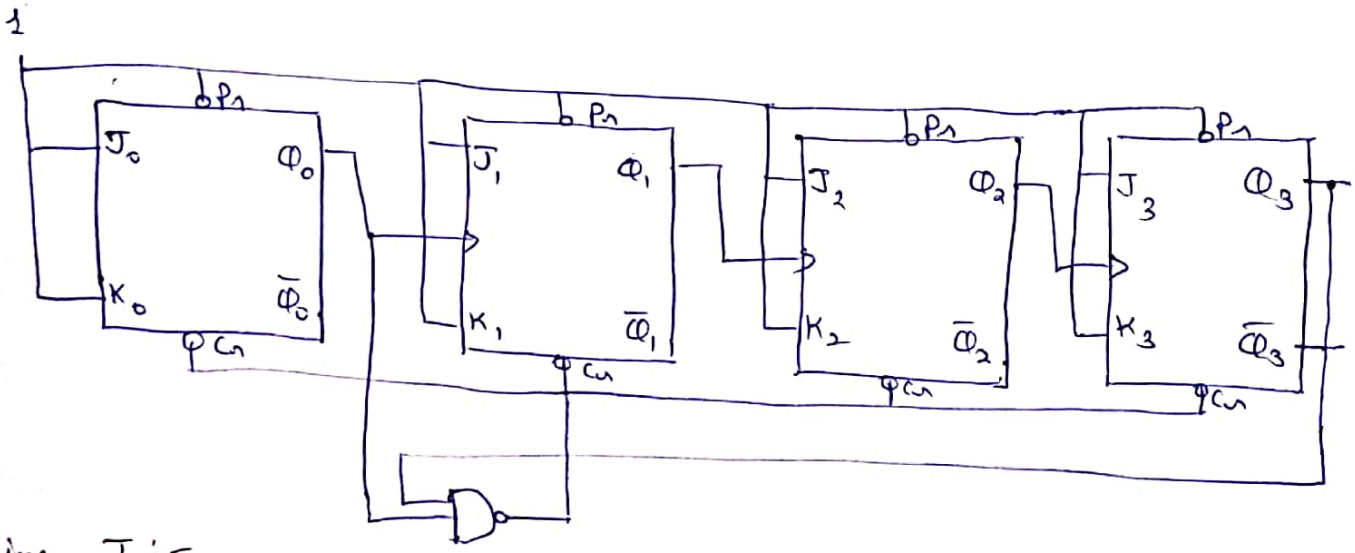
Clock	F/F States				Output.
	Q_3	Q_2	Q_1	Q_0	Y
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10					X
11					X
.					.
.					.
.					.
15					X



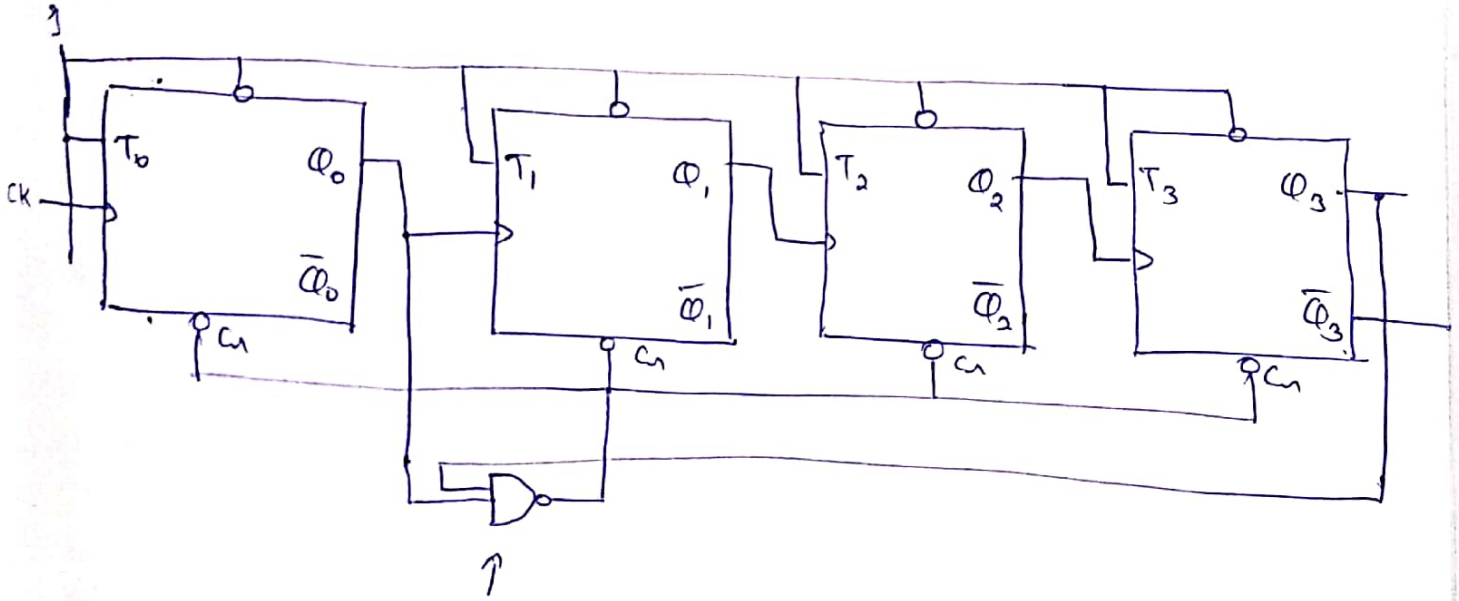
$Y = \bar{Q}_3 + \bar{Q}_1\bar{Q}_0$

$= \overline{Q_3 \cdot Q_0}$

(5) Using JK



Using T :-



It can also be designed by taking $\bar{Q}_3$ & $\bar{Q}_0$ in an OR gate.

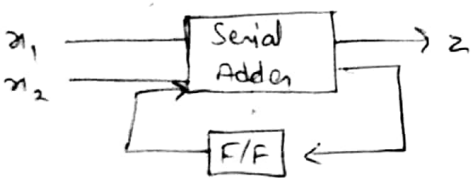
Synthesis of synchronous Sequential Circuits.

Steps are :-

- Form description of the problem through state table/diagram.
- If possible, reduce states.
- Make state assignment & derive transition table.
- Select type & no. of memories.
- Use K-map to obtain simplified f/f inputs.
- Draw logic or circuits.

↳ Serial binary adder.
↳ Sequence generation
↳ Sequ...

Design of serial binary adder.



- State A represents the state when previously gen. carry is 0 & B represents the state when prev. gen. carry is 1.
- Inputs will be x_1 & x_2, & states will be A & B

Cases:-

(I) When the machine is in state A (Carry from prev. adder is 0)

(i) when $x_1 = 0$, $x_2 = 0$

Sum = 0 ; output = 0

Carry = 0.

(ii) when $x_1 = 0$, $x_2 = 1$

Sum = 1 ; output = 1

Carry = 0

(iii) when $x_1 = 1$, $x_2 = 0$

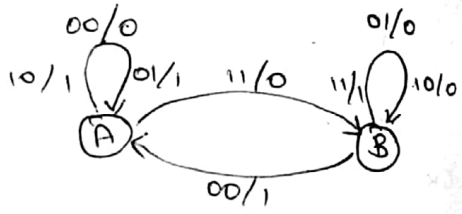
Sum = 1 ; output = 1

Carry = 0

(iv) when $x_1 = 1$, $x_2 = 1$

Sum = 0, output = 0

Carry = 1



state - Diagram

(II) When the machine is in state B! - (Carry = 1)

(i) when $x_1 = 0$, $x_2 = 0$, then

Sum = 1, O/p = 1, Carry = 0, (transition to state A)

(ii) when $x_1 = 0$, $x_2 = 1$, then

Sum = 0, O/p = 0, Carry = 1. (B)

(iii) when $x_1 = 1$, $x_2 = 0$, then

Sum = O/p = 0, Carry = 1 (remain in B)

(iv) when $x_1 = 1$, $x_2 = 1$, then

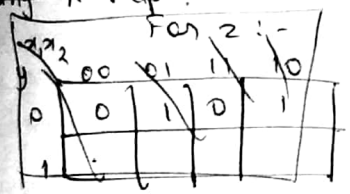
Sum = O/p = 1, Carry = 1 (remain in B)

State - Table

Present - state (y)	Inputs x_1	x_2	Next - state (Y)	Output z	Input to D flip-flop D
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	1	1

Using K-Map :-

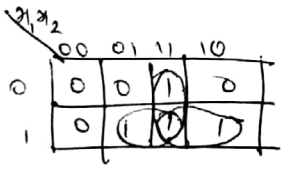
For z :-



$$z = (y \oplus x_1)x_2 + \bar{y}(x_1 \oplus x_2) + y\overline{(x_1 \oplus x_2)}$$

$$= y \oplus x_1 \oplus x_2$$

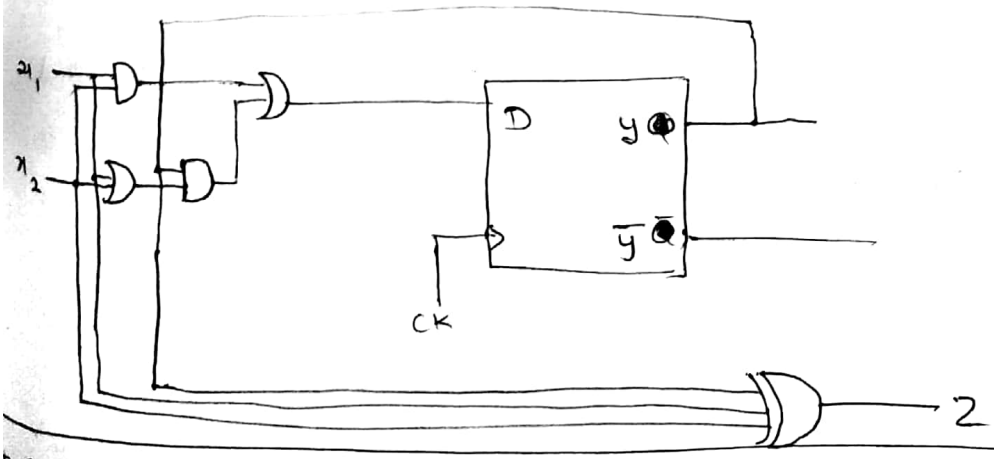
For D :-



$$D = x_1x_2 + yx_2 + yx_1$$

$$= x_1x_2 + y(x_1 + x_2)$$

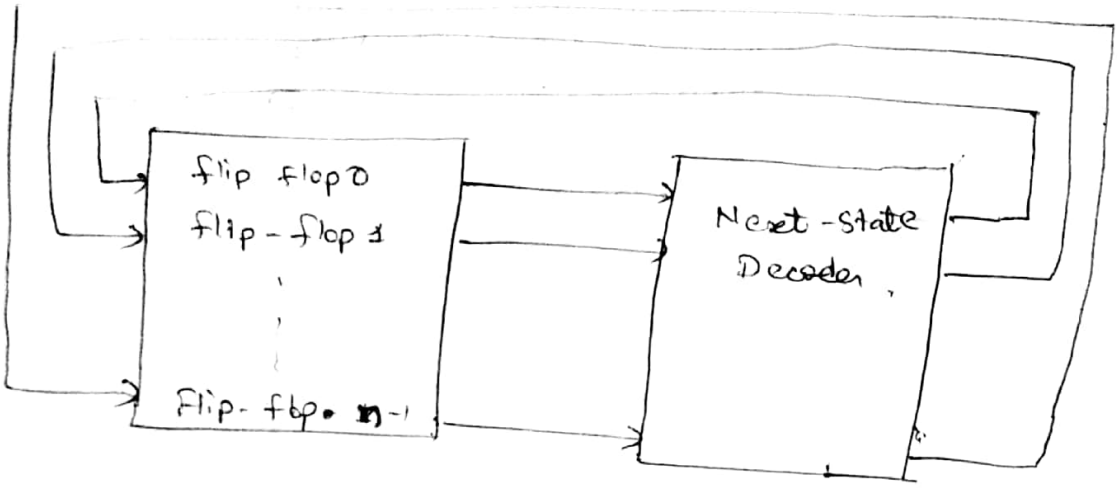
Logic - Diagram :-



Sequence - Generator :-

- Sequential circuit to generate desired sequence of bits in synchronous with a clock.
- Also called pulse generator, in which a particular repetitive series of pulses are generated.

Block - diagram :-



(1) Min. no. of flip - flops(n) : max $(C_0, C_1) \leq 2^{n-1}$

where, C_0 : no. of zeroes in sequence

C_1 : no. of ones in sequence

(2) Draw state - table :-

→ Starting from LSB, assign the desired sequence to the output of the flip - flops,

→ Assign output of flip - flops 0 or 1, such that all states are different.

(3) Draw - state - diagram.

Eg:- Design sequence gen. using D flip flops to generate sequence : 10110011 0.

Solution :-

(1) $\max (C_0, C_1) \leq 2^{n-1}$

$\max(4, 5) \leq 2^{n-1}$

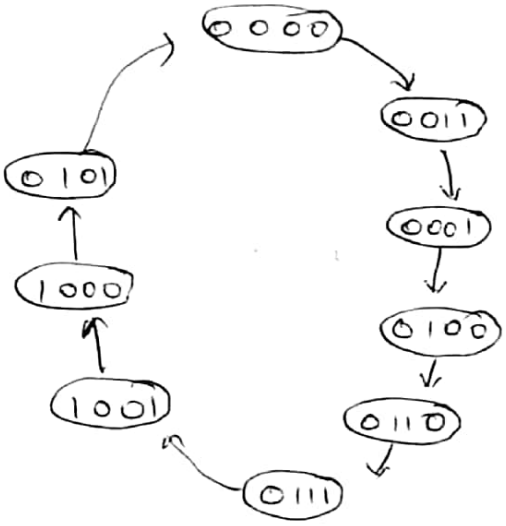
$\Rightarrow 5 \leq 2^{n-1}$

$\Rightarrow \boxed{n = 4}$

(2) State - Table to generate 10110011O :-

Q_3	Q_2	Q_1	Q_0	State
0	0	0	0	0
0	0	1	1	3
0	0	0	1	1
0	1	0	0	4
0	1	1	0	6
0	1	1	1	7
1	0	0	1	9
1	0	0	0	8
0	1	0	1	5

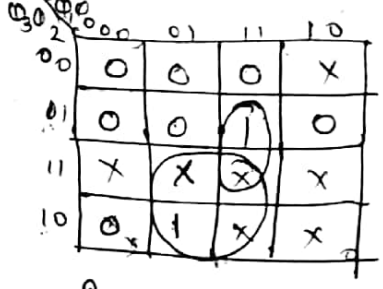
Assigned randomly such that all states are unique.



(3) Final State - Table

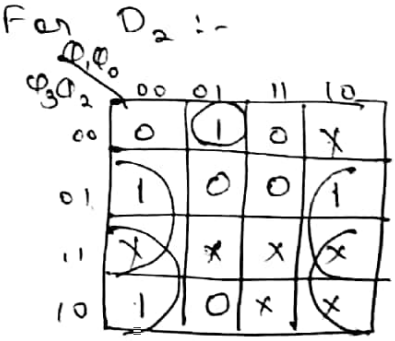
Present - state				Next - state				Inputs to F/F			
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	1	1	0	0	1	1
0	0	1	1	0	0	0	1	0	0	0	1
0	0	0	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	0	1	0	0	0
1	0	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	0	0	0	0	0	0	0

For D_3 :-



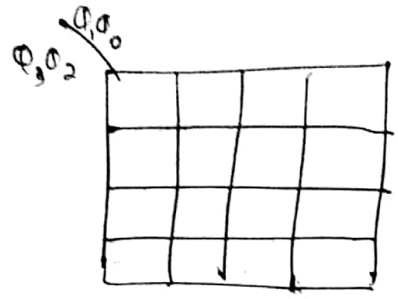
$\therefore D_3 = Q_2 Q_1 Q_0 + Q_3 Q_0$

For D_2 :-



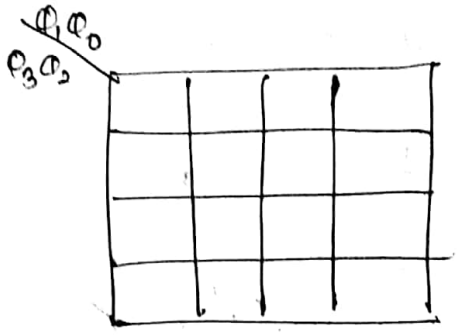
$D_2 = Q_2 \bar{Q}_0 + Q_3 \bar{Q}_0 + \bar{Q}_3 \bar{Q}_2 \bar{Q}_1 Q_0$

For D_1 :-



$D_1 = \bar{Q}_3 \bar{Q}_2$

For D_0 :-



$D_0 = Q_1 + \bar{Q}_0 \bar{Q}_2$

0 0	0 x
0 1	1 x
1 0	x 1
1 1	x 0

• → Draw logic - Diagram

Eg:- Design a sequence-generator using JK : 11001011

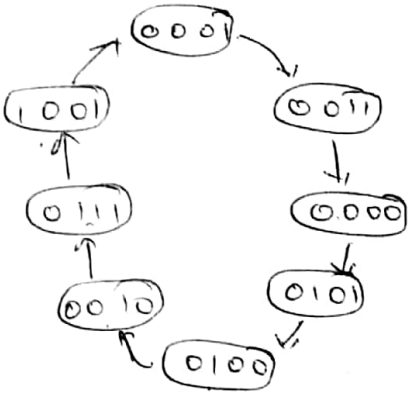
Solution:-

(1) $\max(C_0, C_1) \leq 2^{n-1}$ ⇒ $\max(3, 5) \leq 2^{n-1}$ ⇒ $\boxed{n = 4}$

(2)

Q_3	Q_2	Q_1	Q_0	state
0	0	0	1	1
0	0	1	1	3
0	0	0	0	0
0	1	0	1	5
0	1	0	0	4
0	0	1	0	2
0	1	1	1	7
1	0	0	1	9

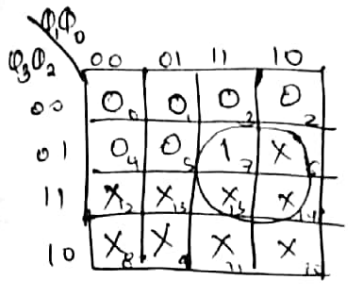
State - Diagram :-



(3) Elaborated State - Table :

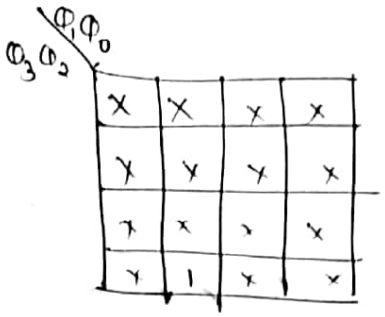
Present - State				Next-state				F/F Inputs							
	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
1	0	0	0	1	0	0	1	1	0	x	0	x	1	x	x	0
3	0	0	1	1	0	0	0	0	0	x	0	x	x	1	x	1
0	0	0	0	0	0	1	0	1	0	x	1	x	0	x	1	x
5	0	1	0	1	0	1	0	0	0	x	x	1	0	x	x	1
4	0	1	0	0	0	0	1	0	0	x	x	1	1	x	0	x
2	0	0	1	0	0	1	1	1	0	x	1	x	x	0	1	x
7	0	1	1	1	1	0	0	1	1	x	x	1	x	1	x	0
9	1	0	0	1	0	0	0	1	x	1	0	x	0	x	x	0

For J_3 :-



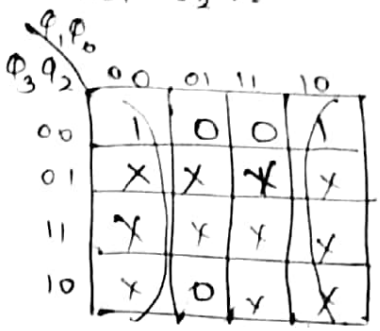
$J_3 = Q_2 Q_1$

For K_3 :-



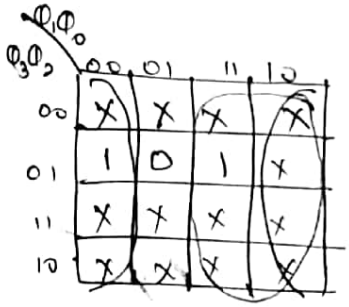
$K_3 = 1$

For J_2 :-



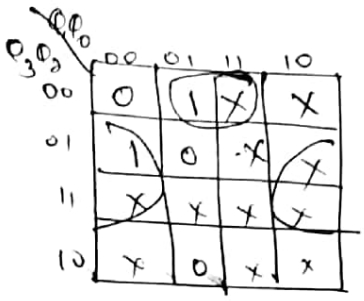
$J_2 = \bar{Q}_0$

For K_2 :-



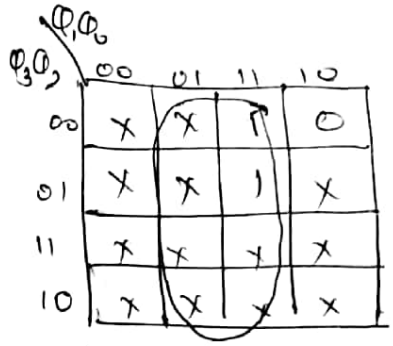
$K_2 = \bar{Q}_0 + Q_1$

For J_1 :-



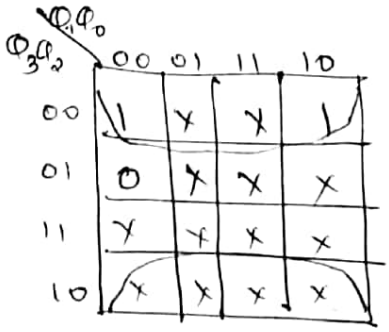
$J_1 = Q_2 \bar{Q}_0 + \bar{Q}_3 \bar{Q}_2 Q_0$

For K_1 :-



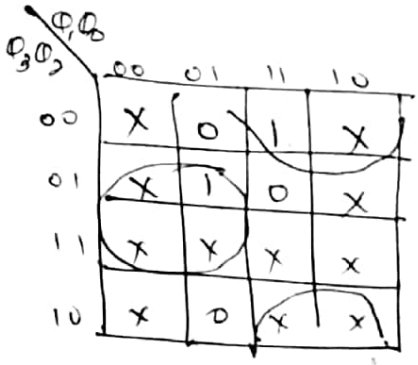
$K_1 = Q_0$

For J_0 :-



$J_0 = \bar{Q}_2$

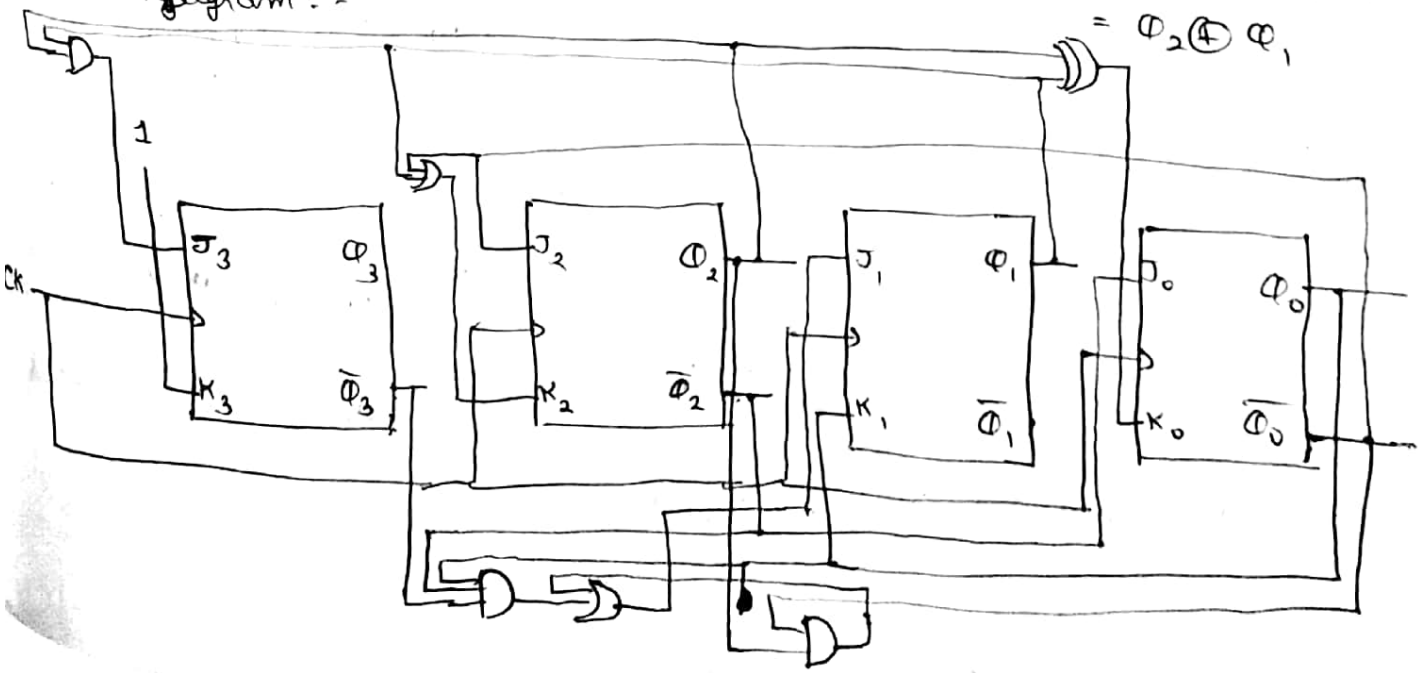
For K_0 :-



$K_0 = \bar{Q}_2 Q_1 + Q_2 \bar{Q}_1$

$= Q_2 \oplus Q_1$

Logic-Diagram :-



Sequence - detector

→

Eg:- Design detector to detect sequence 1010 & overlapping is permitted. Assume the input sequence 01101010 & corresponding output sequence is :.
00000101

Solution:-

No. of bits in sequence = 4 (1010)

So, No. of states = 4 (say A, B, C, D)
1 0 1 0

(i) Present - state is A :-

If input is 0 ⇒ remain in state A & output 0,

If input is 1 ⇒ $A \xrightarrow{1/0} B$

(ii) Present - state is B :- (Previous state was 1)

If input is 0, ⇒ $B \xrightarrow{0/0} C$
1 0

If input is 1, ⇒ $B \xrightarrow{1/0} A$ (possible when overlapping is not permitted).
1 1

$B \xrightarrow{1/0} B$ (if overlapping is permitted).

1 0 1 0

(iii) Present - state is C :- (previous 2 states are 10)

If input is 0 ⇒ $C \xrightarrow{0/0} B$ (Overlapping permitted)

$C \xrightarrow{0/0} A$ (Overlapping not permitted)

If input is 1 ⇒ $C \xrightarrow{1/0} D$

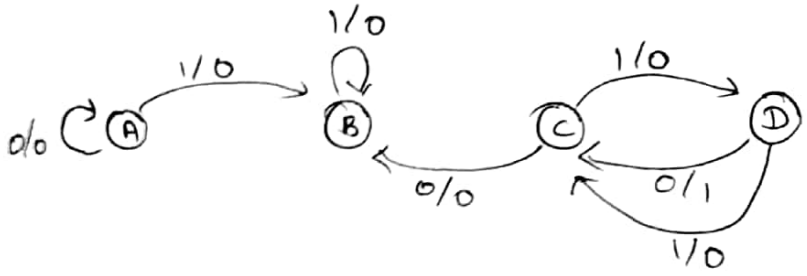
(iv) If present-state is D, (Previous bits are 1 0 1)

If input is 0, $D \xrightarrow{0/1} C$ [Overlapping ✓]

$D \xrightarrow{0/1} A$ [Overlapping X]

If input is 1,

$D \xrightarrow{1/0} C$

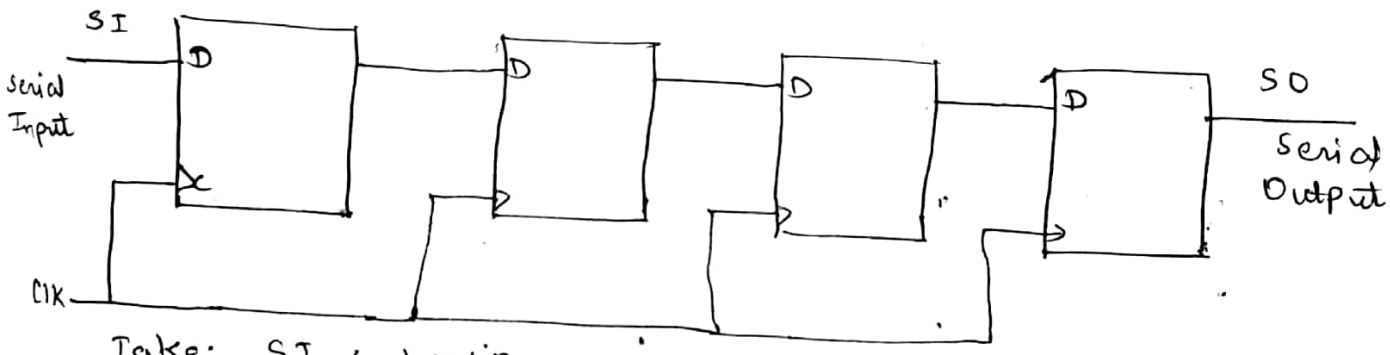


Present - State		Input	Output	Next State		Inputs of F/F	
Y_1	Y_2	x(t)	(z)	Y_1	Y_2	D_1	D_2
0	0	0					
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

→ Shift - Register ←

- A register capable of shifting its binary information in one or both direction is called shift register.
- All clock - pulses receive common - clock - pulse which activates shift registes from one stage to other.

Eg:-



Take: SI : 1 0 1 0

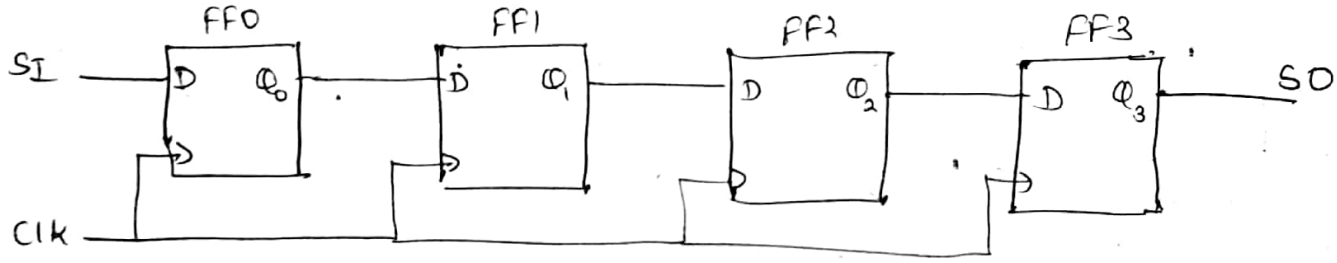
Clk	Q_3	Q_2	Q_1	Q_0	
0	0	0	0	0	
1	1	0	0	0	→ 4 clock-pulse for reading.
2	0	1	0	0	
3	1	0	1	0	
4	1				

∴) For loading (after getting desired SI in ~~o/p~~ of F/F)

Clk	Q_3	Q_2	Q_1	Q_0	O/P	
0	1	0	1	0	~~0~~	
1	0	1	0	1	↑ 0	
2	0	0	1	0	1	⇒ 4 clock pulse for loading.
3	0	0	0	1	↓ 0	
4	0	0	0	0	1	

- Types :-
  - Serial-in, serial-out
  - Serial-in, parallel-out.
  - Parallel-in, serial-out
  - Parallel-in, parallel out.
- Bidirectional Shift Registers.
- Special :- Shift Counters
  - Ring - Counters
  - Johnsonn - Counters

(a) Serial-in, serial out :-



For Rest, see slides.

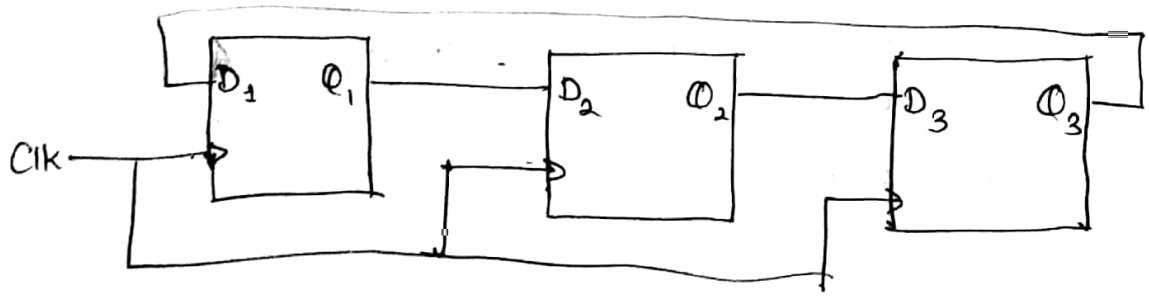
Shift register counters :-

→ Ring - Counter (basic/ simple)

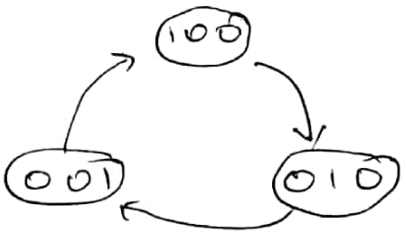
→ Johnson Counter (twisted ring / Switch tail)

Ring - Counter

3 bit synchronous ring - counter using D F/Fs.



After CLK	Q_1	Q_2	Q_3
0	1	0	0
1	0	1	0
2	0	0	1
3	1	0	0



∴ It is a mod-3 counter [∵ No. of states = 3]

Note:-

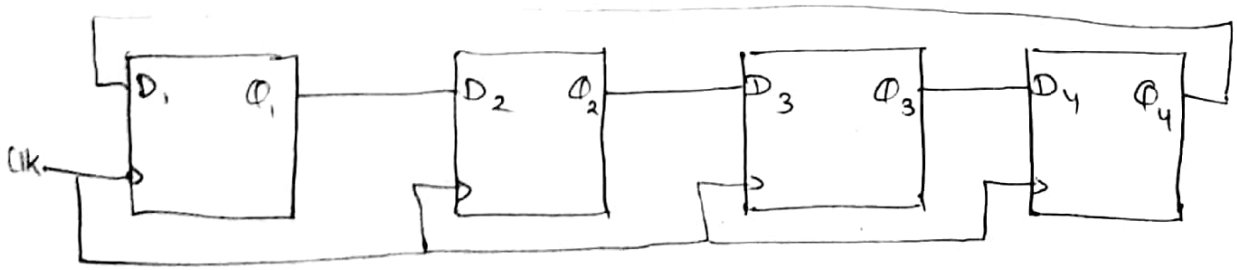
In Ring counter, 3 F/Fs ⇒ States = 3, mod = 3

In Binary counter, 3 F/Fs ⇒ States = 8, mod = 8

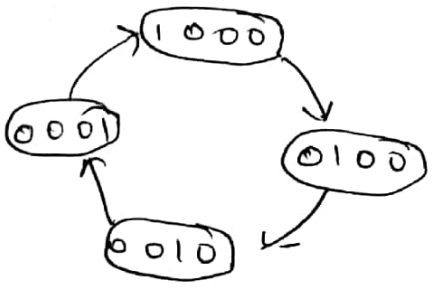
* mod-n ring counter requires n Flip-Flop's connected in a feedback way. So, A ring counter requires more flip-flops than binary counter of same modulus (i.e., mod-n).
* But, Ring-counter is self-decoding. Hence, it is used widely.

Eg:- 4 bit synchronous ring-counter (or shift register) & find modulus.

Solution:-



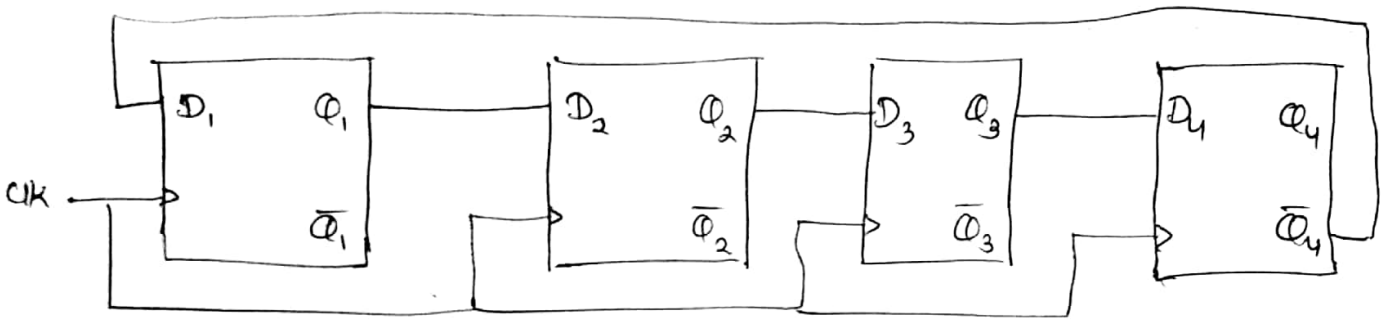
After CLK	Q_1	Q_2	Q_3	Q_4
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0



∴ It is mod-4 counter,

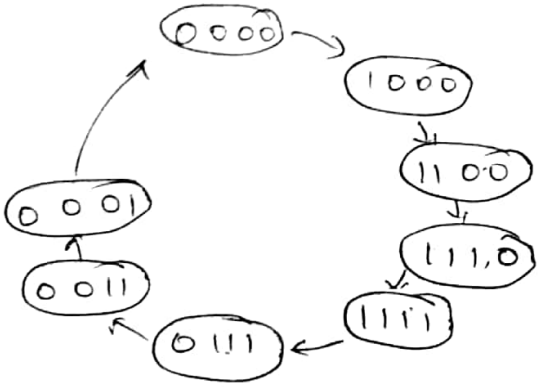
Johnsonn - Counter.

Eg:- 4 bit Johnson Counter using D F/Fs.



After Clk	Q_1	Q_2	Q_3	Q_4
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0

* A n flip-flop Johnsonn Counter have '2n' no. of unique states & it can count up to 2n no. of pulses.
→ So, It is mod(2n) counter.



* # Comparision b/w Binary, Ring & Johnsonn - counters:-

→ No. of flip-flops:- Binary < Johnsonn < Ring.

→ Decoding Circuit:- Ring < Johnsonn < Binary.
(self-decoding)